MASTER IN COMPUTER SCIENCE

On the Design of an Image processing Tool To Help Cell Enumeration

Liteanu, Ariane

Award date:
2018

Awarding institution:
University of Namur

University of Namur
Faculty of Computer Science
Academic Year 2017–2018

On the Design of an Image Processing Tool To Help Cell Enumeration

Ariane Liteanu

Supervisor: _____ (Signed for Release Approval - Study Rules art. 40)
Professor Jean-Marie Jacquet

A thesis submitted in the partial fulfillment of the requirements
for the degree of Master of Computer Science at the Université of Namur

Abstract

Automating the cell detection and enumeration through a picture is a way to save time for biologists during their experiments. However, if the cells' background is noisy, e.g. the surface is porous, the task can become much more complex. This thesis presents an algorithm combining filters and the morphological operations of erosion and dilation to isolate cells with a visible nucleus and count them. This is done in two stages: first, separating the cells from their background, then isolating nucleoli, distinctive part of the cell's nucleus. Mathematical morphology allows to ignore intensity variation in images and help to remove noise elements. Identified cells are notified on the original picture and their number is given to the user. This processing is done by means of Python scripts. In order to improve the usability for people unfamiliar with programming, several interface solutions were analysed. Based on prototypes, their advantages and disadvantages were highlighted.

Résumé

Automatiser la détection et l'énumération de cellules via leur photo est un moyen de faire gagner du temps aux biologistes lors de leurs expérimentations. Cependant, la perturbation de l'environnement de ces cellules, comme une surface poreuse, peut considérablement complexifier la tâche. Cette thèse présente un algorithme combinant des filtres et les opérations morphologiques d'érosion et de dilatation afin d'isoler les cellules dont le noyau est suffisamment visible et les compter. Ceci est obtenu en deux étapes : l'isolation des cellules de leur arrière-plan et l'isolation des nucléoles, partie reconnaissable du noyau. La morphologie mathématique permet de passer outre les variations d'intensité que l'on retrouve sur les images et aide à éliminer les éléments perturbateurs de l'image. Les cellules identifiées sont alors notifiées sur l'image d'origine et leur nombre est communiqué à l'utilisateur. Ces traitements sont effectués grâce à des scripts Python. Afin de rendre leur utilisation plus aisée pour une personne peu familière avec la programmation, plusieurs solutions d'interfaces ont été analysées. Sur base de prototypes, leurs qualités et défauts ont été mis en évidence.

Acknowledgement

Alors que je termine la rédaction de cette thèse, je pense qu'il est temps de donner quelques remerciements. Tout ceci n'aurait pas été possible sans mon promoteur, le Professeur Jean-Marie Jacquet. Je voudrais le remercier pour ses conseils et son soutien tout au long de ce travail. Merci à Marie Fourrez pour sa coopération et les informations au sujet des images de cellules.
Je pense qu'une fois de plus, je me dois de remercier David pour sa patience et sa compréhension devant cette idée un peu folle qu'est de faire un deuxième mémoire.
J'ai pu compter sur mon entourage pour me soutenir dans cette aventure. Grâce aux échanges avec diverses professionnels j'ai beaucoup appris et ça a de la valeur à mes yeux. Merci à Loïc Quertenmont pour avoir pris le temps de partager son expérience.
J'ai également une pensée pour Geoffroy qui a fait le même pari fou. Il est clair que pouvoir partager les hauts et les bas m'a aidé à tenir bon. A tous, merci.

Contents

Chapter 1

Introduction

Scientists from the Faculty of Medicine of the University of Namur are running experiments on colonies of tumorous cells grown on porous membranes as part of their cancer research. The team of Professor J.-P. Gillet studies 17T and 63T cells coming from melanomas. One time consuming part of the research consists in counting all cells that successfully crossed the porous membrane, i.e. cells whose nucleus is visible.

Even if cells have recognizable features, this task cannot be performed by the commonly used software, *ImageJ*. Indeed, it is unable to differentiate the cells from the pores of the background. Consequently, the team has to manually count each cell, picture by picture, on a computer. Using GIMP, they put a red dot on each counted cell. A process that is repetitive, tiresome and time-consuming. Not to mention that is it easy to make mistakes.

The objective of this master thesis is to propose a solution to assist biologists in the counting activity. The first solution shall be adapted to the pictures supplied by the Faculty of Medicine. Then reflection could be opened to the application of the solution to a larger set of pictures.

This document is organised as follows. The characteristics of an image and their utility are presented in Chapter 2. Focus is brought to the specificities of cells enumeration. In order to tackle this problematic, two axes are identified:

- creation of an algorithm which aims at detecting and counting cells on an image.
- implementation of a graphic interface as an intuitive and user-friendly tool.

Three chapters are dedicated to the algorithm. Chapter 3 consists in a state of the art of the image processing methods. Chapter 4 details the Mathematical Morphology method, bringing focus on the erosion and dilation operations. Chapter 5 is dedicated to the algorithm proposed and the results obtained. The graphical interface is presented in Chapter 6 and conclusions are drawn in Chapter 7.

Chapter 2

Contextualization

The cell detection problem calls for image processing. This chapter recalls the basis of both images and cells.

2.1 What is an image?

An **image** is the visual representation of an object, a scene or a concept [1]. It might also have a much abstract signification, mostly the result of human imagination. A picture is the most general term. It can be a painting, a pencil drawing or digital. This is the aspect used in this thesis.
Sometimes abbreviated as pic, a picture is a visual capture of an object. Pictures can be created using devices such as a digital camera, scanner or smartphone (photographs) [2]. Those visual objects could also be made exclusively on a computer and not captured by another device (clip arts, render, illustrations).

2.1.1 Digital images

An image can be considered as digital if it has been acquired, created, processed or stored as a binary file [4]. It is a multi-dimensional signal [3]. Most of the time, this signal is a measure of light, like in photos. But it could also be a representation of density (in x-rays), water concentration (MIR) or metabolism (positron emission tomography).

2.1.2 Bitmap versus Vector images

There is two different kinds of technology that are used to render a visual appearance. *Bitmap* (also known as raster) and *Vector* images.

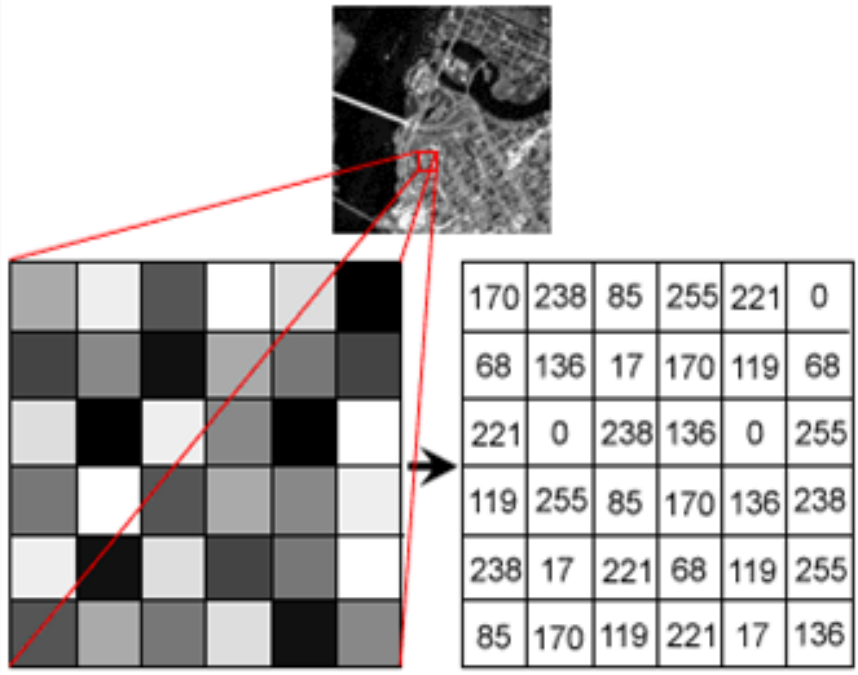


Figure 2.1: Illustration of the quantized nature of digital images [3]. A 2D image is composed of tiny pixels which contain a specific signal value.

Figure 2.2: Comparison of a raster image (left) and a vector image (right). Vector images can be scaled up without losing quality [4].

The Bitmap is closer to our common vision of an image. It is a collection of dots organised in space. Like a printer would place a specific dot of ink in each spot of a sheet of paper, the digital representation of the image is made of **pixels** [5].
Each pixel is therefore a tiny square with a well defined position in a matrix and its own colour (see Figure 2.1). This is the technique used for scanners and cameras. By arranging pixels and slowly incrementing or changing the color or shade of the pixels adjacent to them, it creates a subtle gradation from one color to another [6]. However, if the image file is enlarged without changing the number of pixels, the image will look blurry, as illustrated in Figure 2.2.
The common extensions are .bmp, .gif, .jpg, .png. They can easily be edited using an image or photo editor [2].

The vector image does not use static pixels but points, lines, and curves that are based upon mathematical equations. This means that even scaled-up, the visual appearance will stay the same. A vector graphic is created using dedicated software [6] and its extension will depend on it. `.ai` is from Adobe Illustrator and `.odg` from the OpenOffice tool, Draw.

The table below inspired from [6] summarizes the differences between a Bitmap (raster) and Vector image.

Bitmap	Vector
Pixel-based	Shapes based on mathematical calculations
Best for editing photos and creating continuous tone images with soft color blends	Best for creating logos, drawings and illustrations, technical drawings.
Do not scale up optimally - Image must be created/scanned at the desired usage size or larger	Can be scaled to any size without losing quality. Resolution-independent: can be printed at any size/resolution
Large dimensions and detailed images equal large file size	A large dimension vector graphic maintains a small file size
More difficult to print using a limited amount of spot colors	Number of colors can be easily increased or reduced to adjust printing budget
Depending on the complexity of the image, conversion to vector may be time consuming	Vector art can be used for many processes and easily rasterized to be used for all processes
Most common image format, including: `.jpg`, `.gif`, `.png`, `.tif`, `.bmp`, `.psd`, `.eps` and `.pdfs` originating from raster programs	Common vector graphic file format: `.ai`, `.cdr`, `.svg`, `.eps` and `.pdfs` originating from vector programs
Common programs: photo editing / paint programs such as Photoshop and Paint Shop, GIMP (free)	Common programs: drawing programs such as Illustrator, CorelDraw, Inkscape (free)

2.1.3 Raster images properties

Digital images are often restricted to what are in fact raster images (the bitmaps). In the literature, the aim of image processing is to extract morphological information. Raster images are the technology of photos and movies and unlike vector graphics, the raw signal does not give direct information about its content. If we can easily recognize an apple on an image, a computer has trouble to find a meaning to that group of pixel. It can give back a series of properties but in therm of meaning, it needs image processing algorithms.

The signal carried in raster images is:

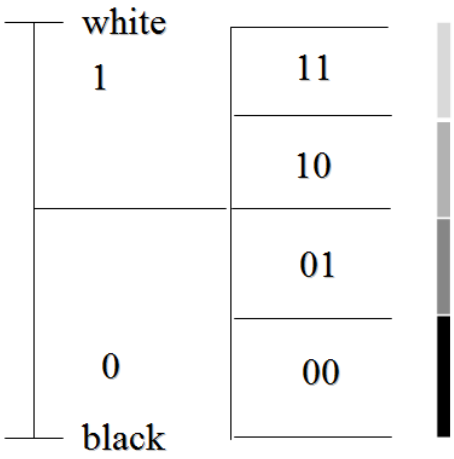
1. Sampled in a limited number of locations
2. Quantized

The first point is about space. A raster image is made of **pixels** (or **voxels** for 3D images) [7]. The second point is about values of the signal. Each pixel will hold a specific

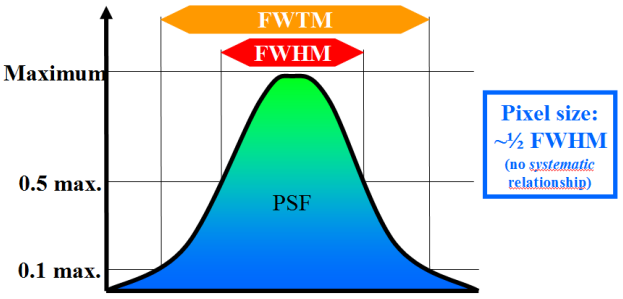
signal. In Figure 2.1, each small box represents a pixel of the image. They are so small that human eye cannot differentiate them [8]. Each pixel contains a signal that must be contained in a range of specific values. The reduction of the signal to one of the possible value in [1,2,3,4,... 2b-1] is called the **quantization**. Here, b is the **bit depth** and is defined by the user or a standard.

The matrix that composes the image has several features [3] :

* a **size**: commonly, images are represented with two dimensions, but it might happen that three or more dimensions are used. Generally speaking, the size is the number of pixel along each dimension axis[1].
* a **bit depth**: number of potential values for each pixel (see Figure 2.3a)
* a **palette**: mapping from pixel value to displayed color
* a **definition**: number of pixels per length unit (600 dpi printer – dot per inch)
* a **point spread function**: measured image of a point object, namely an impulse response function of the imaging device (see Figure 2.3b)
* a **spatial resolution**: full width at half maximum of point spread function (see Figure 2.3b)



(a) Depth of an image [3]. Binary representation: 2 bits, 2^2 levels.



(b) Point spread function and image resolution [3].

Figure 2.3

While *resolution* depends on the image acquisition process, the *size* is specific to the image itself and the *definition* characterizes the image display. The latter is the detail an

[1]If the image is square, otherwise, it is the Cartesian product of those numbers.

image holds. The higher the definition, the heavier the file will be. But this means that it will also be possible to increase the size of the image without loosing too much quality.

Image resolution can be measured in various ways. Resolution quantifies how close lines can be to each other and still be visibly resolved [9]. Therefore, for a given resolution, the pixel size is usually chosen to be slightly smaller than it. Any smaller details would be anyway lost.

How do we define the image bit depth? A common bit depth is 8 which gives a maximum number of 256 colors (2^8) and is widely used to store image information in a computer's memory. To represent coloured images, three 8-bit channels, can be used. Each of them will refer to a primary color, i.e. red, green or blue, and the combination of the three will allow 2^{8*3} colors. This is referred to as 24 bits per pixel (bpp)[10].

A digital image may have a very large depth but the human vision can only discern about 10 million different colors, so saving an image in any more than 24 bpp is excessive. Even if digital imaging allows visualisation in a range of 0,001 nm to 100000nm wavelengths, common picture will only work in the 350 - 700 nm range. However, if the picture is intended for post-processing, a *high-bit* file is recommended to end up with a perfectly preserved gradation of color [10].

Figure 2.3a shows a simple example of a 2 bit depth scale. The possible values are 11, 10, 01, 00 and the gray scale on the right is the *palette*.

In the remaining of this thesis, unless otherwise specified, digital images will be referred as pictures *or simply* images.

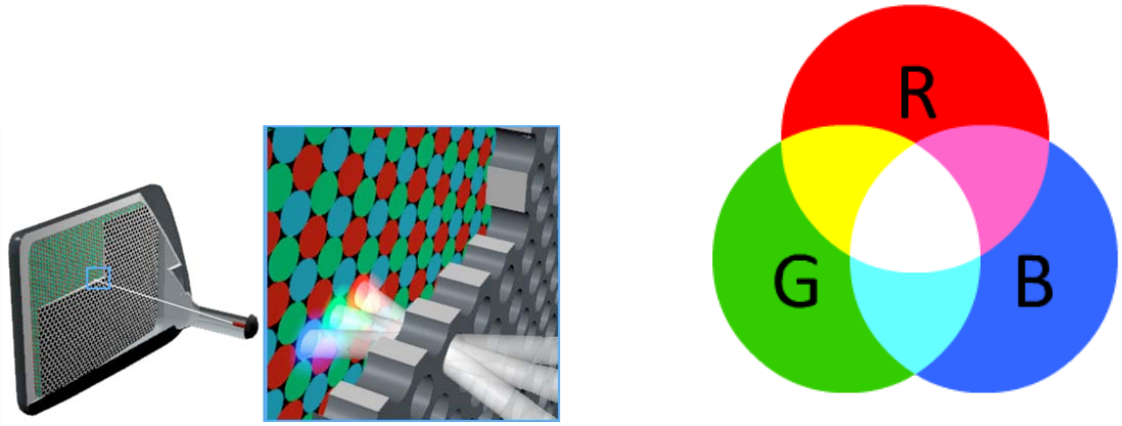
2.1.4 Colours and formalism

Pixels hold the colour information. Sometimes, the colour can be expressed with a single number, like gray scale of binary images. But when it comes to coloured images, different formalism exist. It can be expressed by their names like 'red', 'blue', 'cyan', "magenta" but when dealing with media for screen, the convention is a three channel pixel.

RGB

With RGB value system the three channels correspond to (Red, Green, Blue). They describe the nature of a colour by amounts of each of these colours. Like the old cathodic tv, the colour that is perceived is in fact a combination of three base colours (see Figure 2.4b). This is an additive system, meaning that the more of each color, the brighter (see Figure 2.4a). Even if this is a concise system, it is difficult to distinguish amounts of each channel by eye.

For an 'identical' color that goes darker, the three channels change completely which is not really intuitive. Same thing to describe a hue shift or saturation.



(a) (b)

Figure 2.4: In RGB, the colour is defined by the amount of red, green and blue that it contains. The more of each colour, the brighter the result.

HSV

In HSV, H is for Hue (or color-depth), S for Saturation (or color-purity) and V for Value (or color-brightness) .
This is a colour system that breaks colour down into more simplistic characteristics. Each of its compound (hue, saturation and value) can be adjusted independently in the triplet (H,S,V).

Saturation represents the amount to which that respective color is mixed with white and *value* represents the amount to which that respective color is mixed with black. Both may vary according to the lighting condition of that environment [8]. HSV is a more appropriate approach to perform operations on images [11].

Figure 2.5: In HSV colour system, the hue can be selected independently from its saturation or brightness.

Here are the *hue* ranges for basic colors [12]:

- Red 0-60
- Yellow 60 - 120
- Green 120 - 180
- Cyan 180 - 240
- Blue 240 - 300
- Magenta 300 - 360

Conversion from one to another

RGB is expressed as a triplet of values range between 0 and 255.
HSV is also a triplet but the first value, the *hue*, is expressed in degrees ranging from 0 to 360. The *saturation* and the *value* are percent. Therefore those are numbers from 0 to 100.

The *value* is simply C_{max} and the *saturation*, when this C_{max} is non zero, is equal to the difference of the maximum and minimum values divided by C_{max}.
Calculating the *hue* is a little bit more complex. Equations can be found in Figure 2.7.

$$R' = R/255$$
$$G' = G/255$$
$$B' = B/255$$
$$C_{max} = max(R', G', B')$$
$$C_{min} = min(R', G', B')$$
$$\Delta = C_{max} - C_{min}$$

$$S = \begin{cases} 0 & , C_{max} = 0 \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0 \end{cases}$$

Figure 2.6: Equations for saturation

$$H = \begin{cases} 0^\circ & \Delta = 0 \\ 60^\circ \times \left(\frac{G'-B'}{\Delta} mod 6\right) & , C_{max} = R' \\ 60^\circ \times \left(\frac{B'-R'}{\Delta} + 2\right) & , C_{max} = G' \\ 60^\circ \times \left(\frac{R'-G'}{\Delta} + 4\right) & , C_{max} = B' \end{cases}$$

Figure 2.7: Equations for Hue

2.2 Cells in pictures

" *Cells are the basic building blocks of living things. The human body is composed of trillions of cells, all with their own specialised function* [13]. "

Even if cells may have a variety of shapes and functions, depending on if they are eukariotic or prokariotic cells, they share certain characteristics.
Unlike prokariotes, eukariotes have membrane-encased organelles, including a **nucleus**. The nucleus houses the cell's genetic information contained in its deoxyribonucleic acid (DNA) [14]. All those precious informations are protected by the nuclear envelope. The DNA, along with proteins and RNA[2], forms the **nucleolus**, a circular structure that is found in the nucleus. It does not have a membrane and appears darker on Figure 2.8.

Cells are the subject of numerous studies which concerns their size as well as their interactions or their number. Indeed, the size of a colony indicates a lot about the cell health and reaction to its environment. However, a cell colony might be composed of a huge amount of them. It takes up to three people and several days to analyse complete images. Therefore, we better understand the necessity of developing tool to assist biologist in their counting.

Counting devices exists in a lot of domain. But when it comes to pictures, morphology and patterns becomes predominant. The next chapter highlights some common techniques that allow to process the image and extract important data. The difficulty with living beings is the lack of repetition in their shapes and spacial organisation. Correctly identifying a cell is therefore much more complex than letters in a text. If the objective is to count complete cells, then identifying the nucleus can be enough. For this application on melanoma cells, this is what will be used.

2.3 Conclusion

Cell enumeration is very important for biologists whether that is for cancer research or any other cell study. However, this is a tedious and time consuming work. This is why we would like to automate the counting identification and counting process. Unfortunately, current available methods are not suitable for all situations. This is the reason why it is interesting to try to develop an alternative image processing tool that would address this specific problematic.

Before any implementation, we recalled what an image is, and more precisely, raster images. We also gave the definition of several important properties as the palette the bit-depth and the image definition. The difference between the RGB and the HSV colour formalism has been highlighted. Finally, we defined the nucleus as the element that would be detected in a cell.

For more information on numerical images, see the `http://www.imedias.pro/cours-en-ligne/` website [4].

[2] ribonucleic acid

Figure 2.8: Transmission electron micrograph illustrating the typical appearance of the nucleus in osmium tetroxide fixed material. In this pancreatic acinar cell nucleus, the only prominent feature is the large nucleolus. [15].

Chapter 3

Image Processing Methods

This chapter, presents different image processing algorithms that can be found in the literature.

The scope of this chapter is not to provide a full description of competing methods but rather to give the reader an introduction to the variety of methods available to perform a detection of the region of interest here being, cells. Most of the concept described are commonly used for radiological images (MRI, X-Rays, CT) but their algorithm can be applicable to images from different origins as well.

Feature detection is concerned with determining the presence of some image properties [7]. This thesis assumes that the property (the cells) is already present and focus on its localisation. The *Labelling*, the " *process of assigning a meaningful designation to each region or class* [7] " shall be confirmed by the user after a solution has been provided.

3.1 State of the art

Image segmentation aims at simplifying an image and transform it into an object more meaningful and easier to analyse. For example, dividing an image into homogeneous regions such that points within the same region are similar in nature [16] (i.e in intensity or texture [7]). By assigning a label to every pixel, lines and curves that compose boundaries of objects are revealed. Ideally, those objects correspond to meaningful regions of interest.

Image segmentation plays a key role in medical imaging applications. It facilitates the recognition of regions of interest (like anatomical structures) and allows for the automation or semi-automation of their analysis [7]. Methods for performing segmentations vary widely depending on the specific application and imaging modality. The articles from Dzung Pham [7, 16] identify the eight following categories which are explained hereunder.

1. thresholding approaches

2. region growing approaches
3. classifiers
4. clustering approaches
5. Markov random field models
6. artificial neural networks
7. deformable models
8. atlas-guided approaches

3.1.1 Thresholding

Thresholding, as well as the classifier, clustering, and Markov random field approaches explained in the next sections, is considered as a pixel classification method. This means that a class is assigned to each pixel. It is a simple but effective method that is often used as an initial step in a succession of image processing operations. The partitioning of the image intensities is based on the histogram of the image and the *threshold* is the intensity value that separate the desired classes. Figure 3.1 is showing an example of histogram where the potential threshold is at the valley.
Thresholoding must be associated with other methods. The reasons it is not self-sufficient are that, in its simplest form, only two classes are generated and furthermore, it is not applicable to multi-channel images (like RGB images).

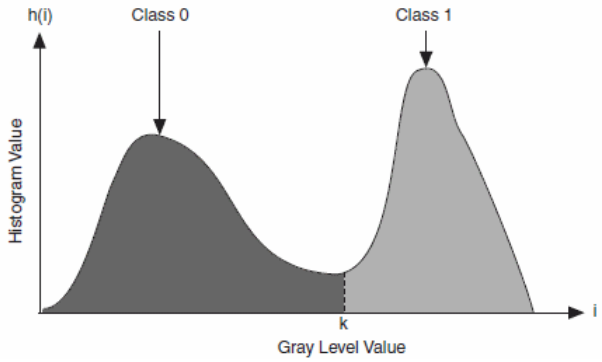


Figure 3.1: Example of histogram with two potential classes of pixels. Axis i represent the gray level value, k is the threshold and h(i) is the number of pixels in the image at each gray level value [17].

3.1.2 Region Growing

In the Region Growing algorithm, *seed points*, strategically placed beforehand, are used to progressively extract the regions having similar properties. This technique typically rely on the homogeneity of the image intensities in the regions of interest. Deeper algorithms which deal with the intensity inhomogeneity that often occurs in real-world images exist [18]. However, the disadvantages remain the same: the seed points need to be manually selected by an operator and number of classes must be known beforehand.
In the case of very small and numerous regions, like cells, requiring a manual interaction might quickly become laborious and time consuming. There are algorithms called "split and merge" [19] which act similar to region growing algorithm but don't require manual definition of point seeds. However, they work with recursive division and merging of the pixels which works best with a gray-scale image like MRI.

3.1.3 Classifiers and Clusters

Classifiers and Clusters perform the same function, the first ones being *supervised* methods and the others *unsupervised*. Supervised means that methods require training data manually segmented and used as references for automatically segmenting new data [7]. The nearest-neighbour classifier is one example of a simple classifier where each pixel or voxel is classified in the same class as the training datum with the closest intensity. With KNN (k-nearest-neighbour), the decision is made considering the k closest values. This is therefore not taking into account any potential statistical structure of the data.
The main disadvantage of classifiers approach is the obtention of training sets which can be time consuming and laborious.

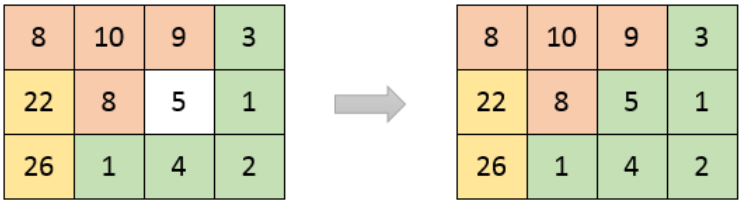


Figure 3.2: Illustration of the nearest-neighbour classifier method. Each pixel has a value (a number) and some of them are already labelled with a class (a colour). The white pixel will join the nearest class which is here the green one with a value of 4 (delta is one).

Sometimes, the class must be identified in a collection of unlabelled data. This problem is addressed by **Clustering algorithms**, one of the most important unsupervised learning problem. A cluster is, in this context, a collection of objects which are *similar* between them and are *dissimilar* to the objects belonging to other clusters [21]. Some common clustering segmentation algorithms are *k-means* [22] (explained hereunder), the *fuzzy c-means* algorithm and the *expectation-maximization* algorithm.

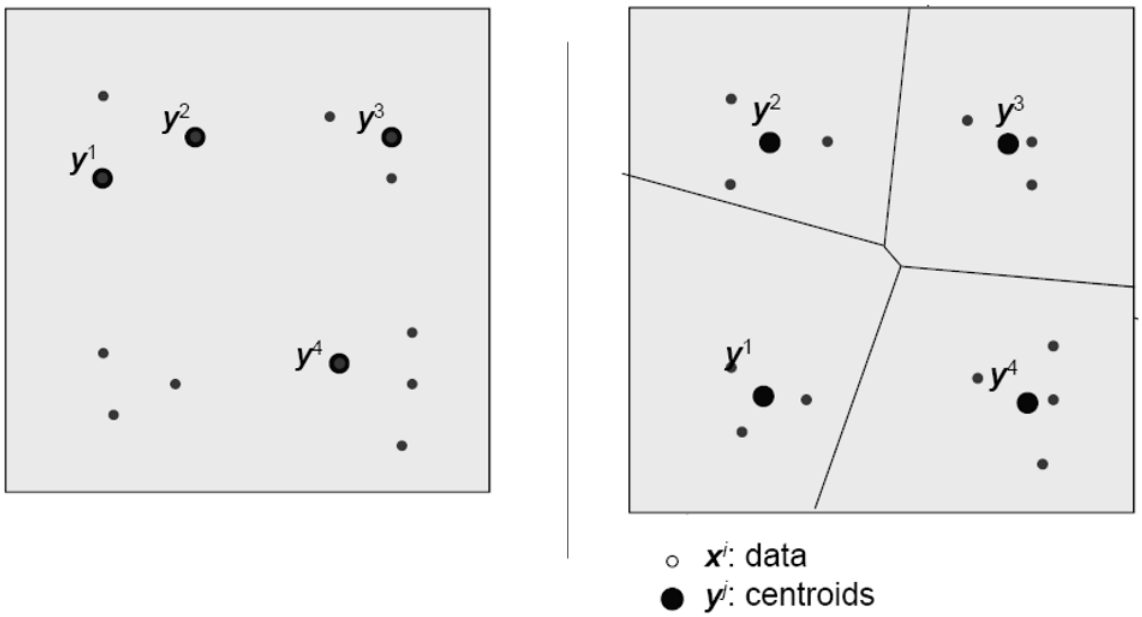


Figure 3.3: Illustration of the k-means algorithm [20]. On the left, the initialization of the centroids (which can be a random selection of points. On the right, after 3 iterations, the centroid have moved to a more stable position and points are gathered in clusters depending on their relative distance.

Clustering can be applied for a variety of fields like:

- Biology: classification of plants given their features
- Economic sciences: identifying distinct groups of customers profile for marketing
- Spacial Data Analysis

The field we pay interest to in the context of this thesis is the image processing. More specifically, the *distance-based clustering* for which the similarity criterion is distance and determine if two objects belongs to the same cluster or not.
By iteration, clustering methods train themselves using the available data but they do require initialization. With the k-means clustering algorithm, after defining a distance function between data points, we need to choose the number of clusters wanted and select k points as starting centroids. The points (or pixel) the closest to a centroid will be assign to its cluster during the 'Assignment step'. Then during the 'Update step' each new centroid location is calculated for each cluster using the mean of all points assigned to this centroid. By iterating over those two steps, centroid slowly come to an optimum meaning that the total distance from each point to its assigned centroid is the lowest. However, this solution is not always the best one as it only provides a local optimum.
Clustering methods lack spatial modelling but can provide significant advantages for fast computation. However, this also means that, like classifiers, clustering algorithms are sensitive to noise and intensity inhomogeneities [7]. One way to decrease the sensibility toward

noise is Markov random field modelling [23] which is a statistical model which can be used within segmentation methods.

3.1.4 Artificial neural networks

Artificial neural networks or ANN are applied for pattern recognition. The parallel networks of processing elements or nodes *learn* by adjusting the interconnection, weights, between layers, and generalizes relevant output for a set of input data [24, 7]. As a paradigm for machine learning they simulate biological learning. Machine learning applied for images is addressed in Subsection 3.1.7. In medical imaging, ANN can be applied mostly as classifiers but also in a unsupervised way as well as for deformable models.

3.1.5 Deformable models

Deformable models are " physically motivated, model-based techniques for delineating region boundaries using closed parametric curves or surfaces that deform under the influence of internal and external forces „ [7]. They give very good results in medical images analysis, like MR images, but they require manual interaction to place an initial model and choose appropriate parameters. Even if these models present robustness to noise, in the context of cell counting, the main concern is finding each cell and the delimitation of their surface is not essential.

3.1.6 Atals-guided approaches

Atals-guided approaches are generally better-suited for segmentation of structures that are stable over the population of study. Which is not the case with the cell counting. The use of statistical atlases in brain images segmentation and the construction of those atlas is detailed in [25].

3.1.7 Machine Learning

Machine Learning is a way to model phenomena in order to take strategic decisions. There are a lot of different machine learning problems but they all are composed of specific elements [26]:

- Datas
- A specific task to be performed
- A learning algorithm
- A performance measure

Given a large amount of data, the algorithm builds an internal representation that enables it to execute the task (for example classifying, prediction, identification) by itself on new data. The first phase is called the learning phase, and the data used for this are part of the training set. Machine learning is based on general algorithm fed with data instead of custom code specific to the problem. The same algorithm fed with different training data will come up with different logic and help the classification. There is not only one learning algorithm to solve all problems. When the algorithm is complex enough, as with artificial neural networks, raw data can be directly analysed. Otherwise, data shall be pre-processed. Usually, the algorithm is chosen depending on the task to be performed (output) and the kind of data available (input). The followings are the most famous ones:

- Linear regression
- K-nn
- neural networks
- random forests

Interest in those techniques is growing rapidly. Even if it can be applied to various kind of data, (logs, user behaviour on a web site, money transactions), machine learning is also used to analyse visual objects. This is called Machine Vision [26, 27]. Its effectiveness does not need to be proven any more, a well known and very convincing example is Google's automatic photo search [28]. Given the huge number of photograph uploaded daily on platforms like Flickr, Facebook or Instagram [1], the learning set has become enormous. Machine learning helps in analysing color and shape patterns, link it to any existing data about the photograph to help the search engine understand what an image actually is.

For example, if the goal is to create an algorithm capable of finding cars on a video of a road, first, a training set is built up from many pieces of pictures that only contains cars. The algorithm will progressively learn what a car looks like and is tested with labelled examples.

One of the main disadvantages of machine learning is that it only works when a lot of data is available for the training part. Furthermore, in this work, the target element that we want to detect are cells. Counting them is one thing, but distinguishing two or more overlapping cells is a way more complicated exercise [29].

3.1.8 Mathematical Morphology

Unlike most of the methods of image segmentation presented above, Mathematical Morphology (MM) is based on shapes (hence the word 'Morphology'). It is a theory for the analysis

[1] over 350 million in 2013 just for Facebook, according to [28]

of spatial structures [30] based on set theory, topology, lattice algebra and random functions (hence 'Mathematical').
It is not a new technique as it was initiated in the late sixties by George Matheron and Jean Serra [30]. MM is not very popular in papers about image processing and computer vision. However, we can find some example of its application to industrial machine vision [31, 32].

Mathematical Morphology's basics arithmetics are erosion and dilation. They are also called "binary morphological operations „since they are applied on binary (black and white) images. Extensions of these relationship have been developed in gray scale morphology [31].

The article [33] shows that extending gray scale MM algorithm to color or vector valued images is complicated as there is no natural ordering on a set of color vectors. However, erosion and dilation are perfectly adapted for the processing of a mask (binary) resulting from a filter.

3.2 Other image manipulations

Here are few other techniques that are not technically speaking segmentation techniques but do have their importance in image manipulation.

3.2.1 Image Histogram

As mentioned earlier, an image is a vector of digital values. Gathering and counting the amount of each value present in an image allows to represent it as an histogram which can then be used to collect information or to modify the image properties.

Adjusting Contrast. The contrast of an image is perceived as the difference between maximum and minimum pixel intensity in it. By analysing the data repartition, we can sometimes observe that useful data populates only a small portion of the available range [3]. By modifying the original values so that more of the available range is used, we increase the contrast. The difference between different elements in the picture becomes more obvious and is easier to process.
There are many different techniques and methods of enhancing contrast and details in an image [34] . Here are few examples:

- Histogram equalization (HE) [35],
- Histogram Matching [34],
- Local Enhancement [34],
- Histogram Stretching [3]

These are *spatial domain techniques* in which pixels values are manipulated to achieve the desired enhancement. However, the image could also be first transferred in the frequency domain by means of a Fourier transform to perform the enhancement operations on the result. This is called *frequency domain methods*.

Figure 3.4 illustrates the "linear stretch" where the original values are stretched so that they populate the whole available range. This technique is often used with x-ray images. Indeed, when imaging high-density object, the beam is said to be attenuated and it results in higher density areas appearing darker. Without a linear stretch, the range of value is narrow and the resulting contrast, low which complicates their interpretation.

Adjusting brightness. Brightness refers to the overall lightness or darkness of the image. Even if it strongly depends on the viewer visual perception, an image brightness can be defined as the amount of energy output by a source of light relative to the source we are comparing it to [35]. The adjustment of brightness can be obtained simply by adding (to increase brightness) or subtracting (to decrease) a number to the whole image matrix. On an histogram, the result is a curve that will be shifted to the left or to the right.

Thresholding. As mentioned earlier, Thresholding is a way to divide an image depending on the content of its pixels. The threshold is the limit above (or under) which the pixels are not retained.

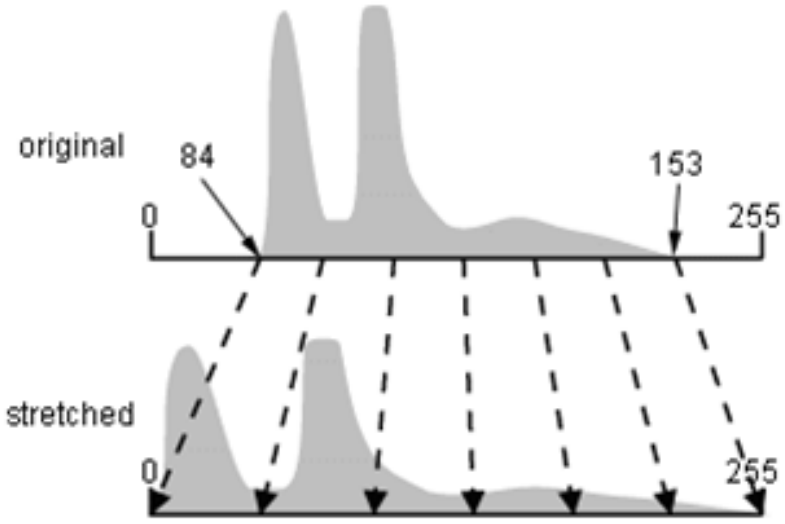


Figure 3.4: A linear stretch involves identifying lower and upper bounds from the histogram and applying a transformation to stretch this range to fill the full range [3]. The image contrast is therefore enhanced.

3.2.2 Filtering

A filter is a mathematical transformation that is used to suppress and/or extract content in images. They are mostly known under the name of **digital filters**. Depending on the filter, one specific information is isolated. This can be compared to an actual coffee filter which will let water go through as well as particles under a certain diameter but will retain all coffee waste.

An image can either be filtered in frequency or in the spacial domain [36, 37]. For the first one, there is an extra step that requires to transform the image into frequency domain to then multiply it by a frequency filter function. The result is finally re-transformed into the spatial domain. The most common function aims at eliminating some of the frequencies from the image. Keeping only high (High-pass filter), low (Low-pass filter) or a selection of frequency (combination of both).

The multiplication in the frequency domain corresponds to convolution in the spatial domain. Most filters are linear and so, based on convolution. The mathematical operation is identical.

The following are common filters (based on a list from [36]):

- Mean Filter - *noise reduction (NR) using mean of neighbourhood*
- Median Filter - *NR using median of neighborhood*
- Gaussian smoothing - *NR using convolution with a Gaussian smoothing kernel*
- Conservative smoothing - *NR using maximum and minimum of neighborhood*
- Frequency filters - *high and low pass image filters, etc*
- Laplacian - *edge detection filter*
- Unsharp filters - *edge enhancement filter*

Edge Filtering. Edge filtering is often cited. It is in fact achieved using a combination of filters as noise filter (Gaussian), gradient-intensity filter and a non-maximum suppression. The Sobel filter is used to emphasis edges. It allows to find the approximate absolute gradient magnitude at each point in an input grayscale image using a set of two convolution kernels; a vertical and a horizontal [3].

Colour Filtering. Colour filtering is a simple form of spacial filtering. Close to the tresholding, boundaries are fixed in the color domain and only the pixels containing the right value are kept.
It may be applied to multi-dimension channels, unlike most of the classic image processing.

3.3 Selected methods

The aim of this work is to identify cells on pictures in order to count them automatically. As such, it is not important to obtain a precise delimitation of the cells. This is a first hypothesis.
A significant problem of our work is that the location of the cells is unknown. Therefore, one of the main goal is to be able to identify the cells without an operator having to manually indicate a starting point in each one of them. This eliminates region growing algorithms, clustering algorithms or deformable models. The lack of labels or stability excludes classifiers or atlas-guided approaches.

Cells can have variable size, shape, the image are multi-channels and they often vary in intensities over the surface. In this context geometrical filters are not suitable as the cells are not regular enough. On the other hand, Mathematical Morphology methods are well suited but the MM methods are more adapted to binary images than to multi channels images. Machine learning algorithm are disregarded as overlapping cells can not be handle easily by such algorithms. Furthermore, they would request a large database which is not available. As for edge-detections, they are not efficient on a picture with noise like pores. This will be illustrated in Chapter 5.

As none of the methods perfectly fits our requirements, they will be combined to reach them. Consequently, the following algorithm is proposed: the picture will be pre-processed with a color filter step and the MM method will then be applied to the preprocessed image to get rid of noises and identify regions of interest. The detail of mathematical morphology is given Chapter 4. The full algorithm implemented will be presented in Chapter 5.

3.4 Conclusion

This Chapter presented a variety of methods available to perform the detection of a region of interest. It was inspired by the article [7] that compare some of them. Most of the time, segmenting an image includes looking at the content of its pixels and gather them with others with a similar nature. Thresholding is quite radical and often used in combination with other techniques. Some exploit already existing information: labels for the classifiers, training sets for machine learning or atlas-based approaches. Region growing, clustering and deformable models will require a starting points and therefore, a manual intervention.
Considering the advantages and disadvantages of each ones, it was decided to base the protocol on mathematical morphology. Combined with the use of a colour filter, an identification of regions of interest (cells) will be possible.

Chapter 4

Mathematical Morphology

Mathematical morphology will be used in our cell detection protocol to get rid of noises and identify regions of interest. In this chapter, the relationships of erosion and dilation will be detailed as well as the tools available in the python library to perform those operations.

4.1 Algorithm

As we focus on the identification of elements composing a picture, shapes have a particular meaning. In digital image processing, Mathematical Morphology is a tool used to investigate the geometric structure. Its goals are to allow the underlying shapes in the image to be identified in spite of imperfections such as, noise, distortions and artefacts. In other words, it will "clean" image data.
According to [31], mathematical morphology is the natural processing approach to deal with the machine vision recognition process. Most morphological operators consist in a combination of primary operations: *dilation* and *erosion*. For the sake of simplicity, we will consider those operators applied on binary images only. If the input image is not a black and white it can be converted by thresholding (for gray-level images), or more specifically in this case, a color filter. We will also limit the analyse to 2D images.

We can represent a binary image as a grid of pixels. Most of them are white (background) and some are black (activated pixel). This is of course depending on the convention. An example can be found in Figure 4.1.a . We often speak of sets. The definition of sets can also be applied to higher dimensional space but in a 2D binary image, it is simply the shapes formed by a same color (for example, all the white points).
One of the set of points is considered as selected. It is those points that are morphologically transformed.

Aside from the original binary image, we also need a second important type of set: the *Structuring Element* (SE).

The structuring element is similar to the image (a grid of pixel in our case), but much smaller [38] and composed of one type of pixel. It also has an *origin* (also called anchor pixel). Depending on how the structuring element will affect the image, the result will be either a dilation or an erosion. The set B in Figure 4.1 is the structuring element.

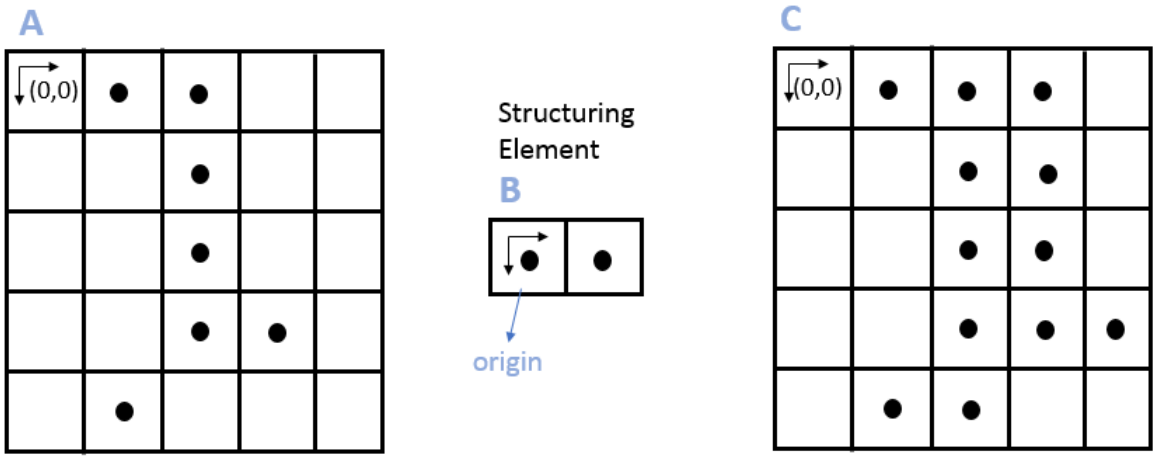


Figure 4.1: Illustration of a dilation applied on image A with Structuring Element B. This result is the image C.

Here under, we will describe the two morphological operations of dilation and erosion.

4.1.1 Dilation

The basic effect of dilation is to gradually *enlarge the boundaries* of regions of foreground pixels [39].
Dilation is the operation of combination. In other words, it combines two sets using vector addition of set elements [31]. This operation is commutative and additive.
It is denoted as:

$$A \oplus B = \{c \in Z^2 \mid c = a + b \text{ for some } a \in A, b \in B\}$$

with $\oplus$ which denotes dilation and the '+' defined as: 0+0=0, 0+1=1, 1+0=1 et 1+1=1

Figure 4.1 represents a dilation operation performed on a simple image of 5 x 5 pixels. Each square with a dot representing a black pixel, others being the white pixels.
The structuring element (a 1 x 2 set) slides over the source image and will "add itself" to the foreground (black pixels on Figure 4.1) each time its anchor pixel meets a foreground pixel (of the source image). In Figure 4.1, the source image A is dilated using the *Structuring Element* B. Image C is the result. We can notice that the pixel (0,3) has turned black. This is the outcome of SE sliding over the image and meet a pixel in (0,2).

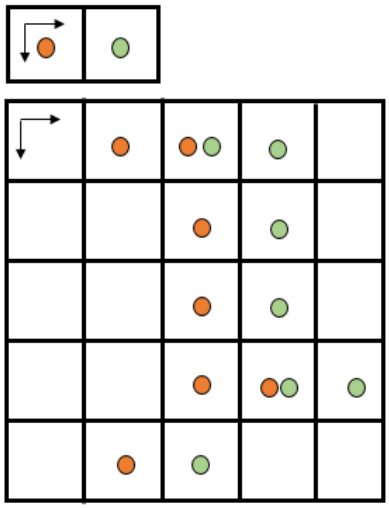


Figure 4.2: Union of translation of A by the elements of B: (0,0) in orange and (0,1) in green.

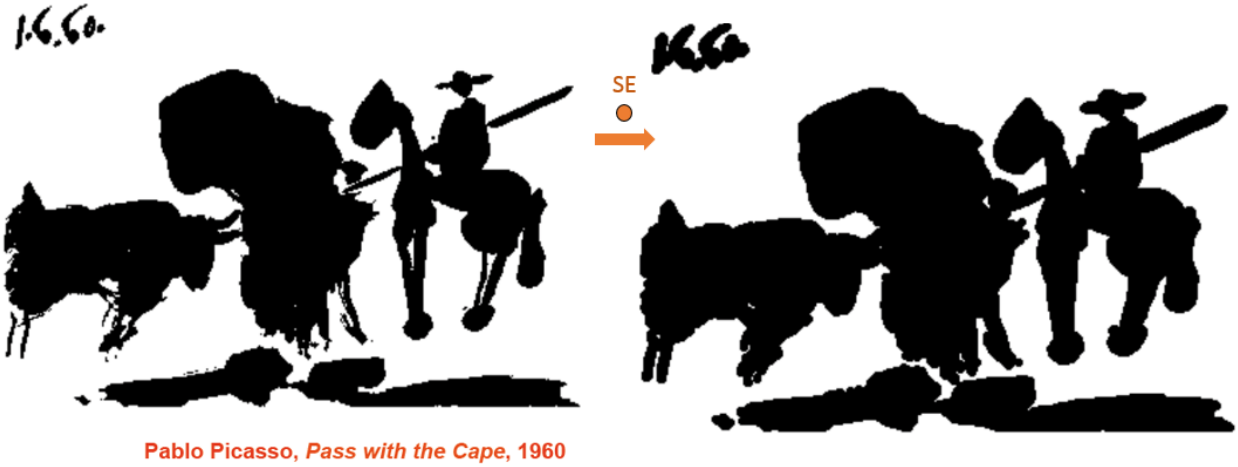


Figure 4.3: Result of a dilation performed on a binary image [3]. The resulting image seems swollen. Some elements (like the legs of the horse) have merged. SE is the structuring element.

It can also be computed as the union of translation of A by the elements of B. Here, those elements are (0,0) and (0,1) meaning that C is the union of the source image (translation of (0,0)) and its translation by vector (0,1) (see Figure 4.2).
After this operation, the image will appear like swollen (see Figure 4.3). Small holes will tend to be filled and separated element will merge.

4.1.2 Erosion

Erosion has the opposite effect of dilation. The Structuring Element also slides over the source image. But here, a foreground pixel is kept only if , once the origin meets a pixel of the foreground, the SE fits completely in it. The subtraction of the SE is illustrated in Figures 4.4 and 4.6. It is denoted as $A \ominus B$ and is defined as:

$$A \ominus B = \{x \in Z^2 \mid \text{ for every } b \in B, \text{ exist an } a \in A \text{ s.t } x = a - b\}$$

Erosion can also be defined in terms of translations and intersections.

$$A \ominus B = \{x \in Z^2 | (B)_x \subseteq A\}$$
$$A \ominus B = \bigcap_{b \in B} (A)_{-b}$$

The notation $(B)_x$ represent the translation of matrix B by $x \in Z^2$ defined as: $\{c \in Z^2 \mid c = a + x \text{ for some } a \in A\}$.

The second equation means that we only keep the intersection of A being translated by vectors of B. With this specific SE (see Figure 4.4), the pixels kept are the ones that appear in both the source image (translation of (0,0)) and its translation by vector (0,1). On eroded images, small isolated elements will tend to disappear and surface with peaks will tend to be smoothed. The extend of the effect depends of the structuring element.

It is important to keep in mind that erosion is NOT the inverse of dilation. In general, successively dilating and eroding an image with the same SE does not produce a null effect. In fact, this is the base of composite operations.

4.1.3 Opening and Closing

In practice, dilations and erosions are usually employed in pairs. The result of dilation and erosion combination is called opening or closing depending on which is applied first. In both cases, the aim is an elimination of specific image details smaller than the structuring element without the global geometric distortion of unsuppressed features [31].
Closing is a dilation followed by an erosion with the same structuring element. $B \circ K =$

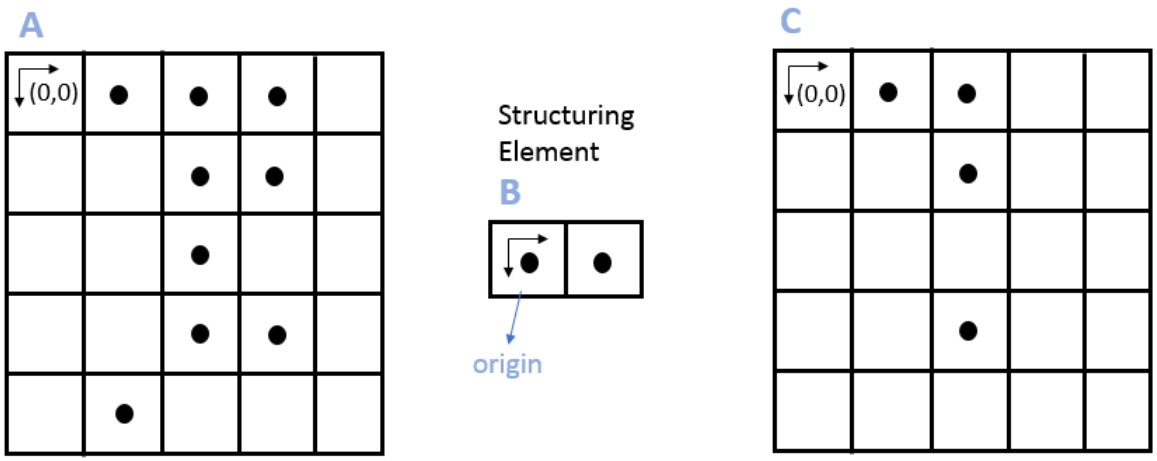


Figure 4.4: Illustration of an erosion applied on image A with Structuring Element B. This result is the image C.

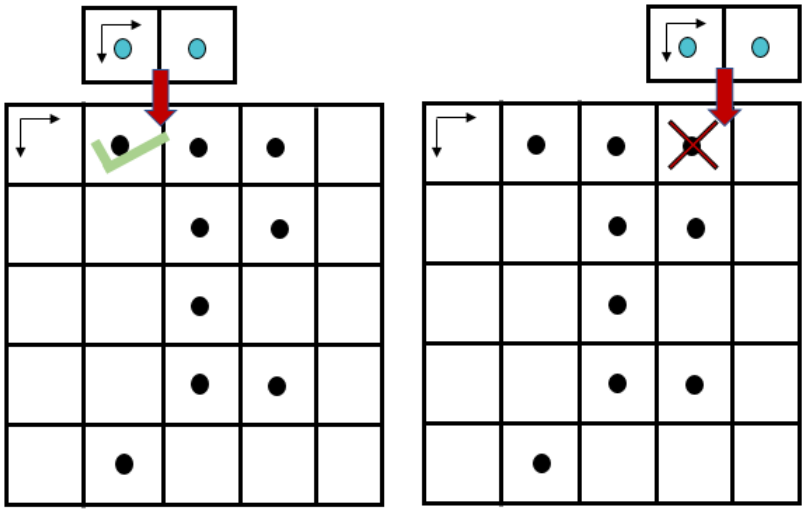


Figure 4.5: Details of an erosion. Pixel (0,1) can be kept as the whole SE can fit in the black area. However, pixel (0,3) will disappear because the SE go beyond the foreground set.

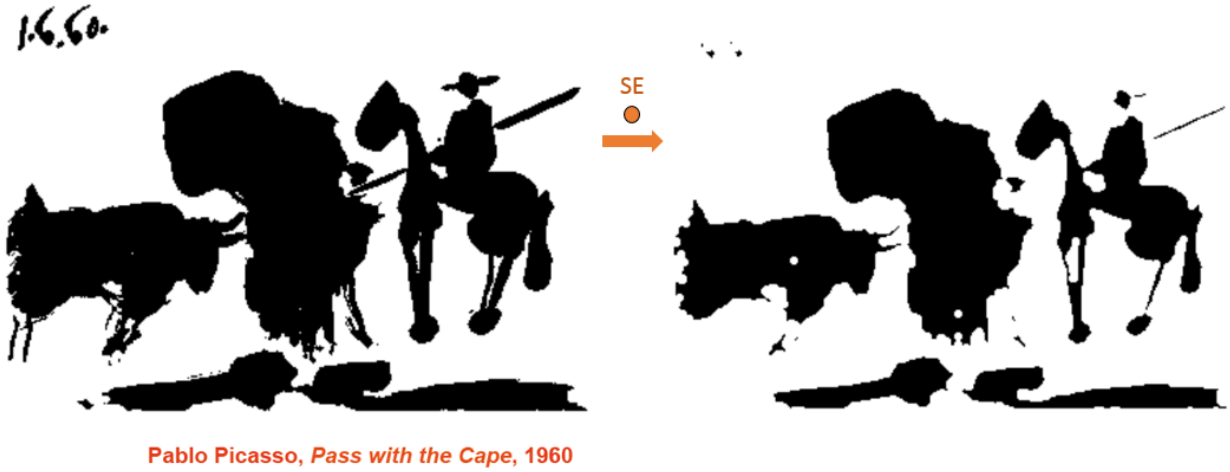


Figure 4.6: Result of an erosion performed on a binary image [3]. The resulting image is smaller. Some thin elements (like the bull's legs) disappeared. SE is the structuring element.

$(B \ominus K) \oplus K$
Opening is an erosion followed by a dilation with the same structuring element. $B \bullet K = (B \oplus K) \ominus K$

An opening have the following effects: smoothed contours, breaks narrow bridges, eliminates small islands. Closing also smoothes but it merges narrow gaps and removes small holes [16].

4.2 Tools

The most popular software to analyse cells is imageJ. It is an open source image processing program designed for scientific multidimensional images [40]. From its website, we can read that it runs on any computer with a Java 1.8 or later virtual machine. Downloadable distributions are available for Windows, Mac OS X and Linux.
This software is however not suitable for cell enumeration on a porous surface. It confuses a cell and a pore. The aim of this thesis is not to explore ImageJ 's features but to offer an alternative to it.

There are many softwares to manipulate pictures in order to extract information from them.
As we mentioned earlier, a digital picture is a matrix of values. Therefore, Matlab, the tool specialized in matrix manipulation, is an obvious choice. And indeed, Matlab possesses a wide toolbox for image processing. Common operation as Fourier transform, filters, or mathematical morphology are already implemented.

The language Java also benefits from libraries to help handling arrays, matrices and pictures. Python however has the advantage of simplicity and code readability. As another major advantage, Python is free and accessible while Matlab is a professional software that has to be purchased. Thanks to its libraries `numpy` , `scipy` or `opencv` (described hereunder), image processing in Python is very similar to Matlab.

Scipy. Scipy is an open-source Python library that is used to define a scientific environment (Matlab, Scilab, R). Scipy uses tables and matrices from NumPy. Example: computes the fast Fourier transform of 'y'

```
from scipy.fftpack import fft
# Number of sample points
N = 600
# sample spacing
T = 1.0 / 800.0
x = np.linspace(0.0, N*T, N)
y = np.sin(50.0 * 2.0*np.pi*x) + 0.5*np.sin(80.0 * 2.0*np.pi*x)
yf = fft(y)
```

Numpy. Numpy is a highly optimized library for numerical operations, adding support for large, multi-dimensional arrays and matrices and the functions to operate on these arrays. Examples: create arrays

```
b = np.array([6, 7, 8])  #creates an array from a regular Python list
SE = np.ones((6,6), np.uint8) #create an 6 x 6 matrix of ones
```

OpenCV-Python. OpenCV-Python is the Python API of OpenCV that currently supports a wide variety of programming languages. OpenCV deals with algorithms related to Computer Vision and Machine Learning. It continues to be expanded and a rich documentation and tutorials can be found on the openCV-python tutorial website [41].
Examples: read in gray scale and dilate an image

```
img = cv2.imread(path_to_image, 0)
res = cv2.dilate(img, SE , iterations=1)
```

res is the result of a dilation of the image 'img' with the structuring element SE.

The combination of the three libraries is considered as an appropriate set-up for fast prototyping of computer vision problems.

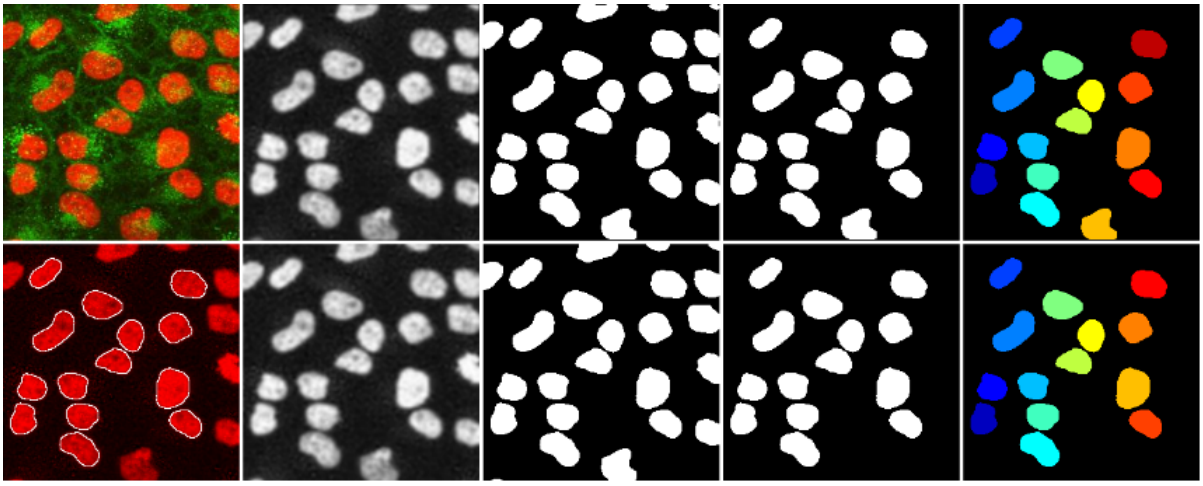


Figure 4.7: Mathmatical Morphology using MATLAB [16]. Processing using spatial filtering, mathematical morphology and segmentation.

4.3 Conclusion

This chapter detailed the mathematical morphology operations of erosion and dilation on binary images. It defined the *Structuring Element* and how it affects the region of foreground pixels. Dilation have an effect that can be expressed as the union of a translation. The resulting image appears swollen. Erosion is the intersection of translations and it tends to make small isolated elements disappear.

The content of this chapter is inspired from the Medical Imaging course of Professors Lee, Janssens and Macq [3] and Mathematical morphology course of Professor Manzanera [42].

Chapter 5

Development: Algorithm

The aim of this thesis is to propose a solution to design a tool helping biologists to count cells from pictures taken under a microscope. This can actually be divided into two sub-problems.

1. The choice of the algorithm processing the image
2. The development of a user interface

This chapter goes through the algorithm development and the next one details the user interface.

5.1 Development of the protocol

As the aim of the thesis is to enumerate cells on a specific type of pictures, the algorithm should identify them and show them to the user. The software would then deliver an approximation of the total number.
From a discussion with Marie Fourrez from UNamur, who is familiar with the experiments with melanoma cells, the ideal error threshold should be of 1 %. If it is a little ambitious, it should at least not exceed 5 %.

5.1.1 Tumour cells on porous membranes

The Faculty of Medicine provided pictures of cell colonies grown on porous membranes. Those pictures are especially large. To ease their processing they have been divided. The sample presented on Figure 5.1 was used for the development of the algorithm whereas the others have been kept for validation.
As part of their experiments, biologists need to count all cells that successfully crossed the

porous membrane. This means the ones for which the nucleus is visible.

Currently, they use a manual counting, picture by picture, on a computer. Using GIMP, they put a red dot on each counted cell which is repetitive and tiresome.

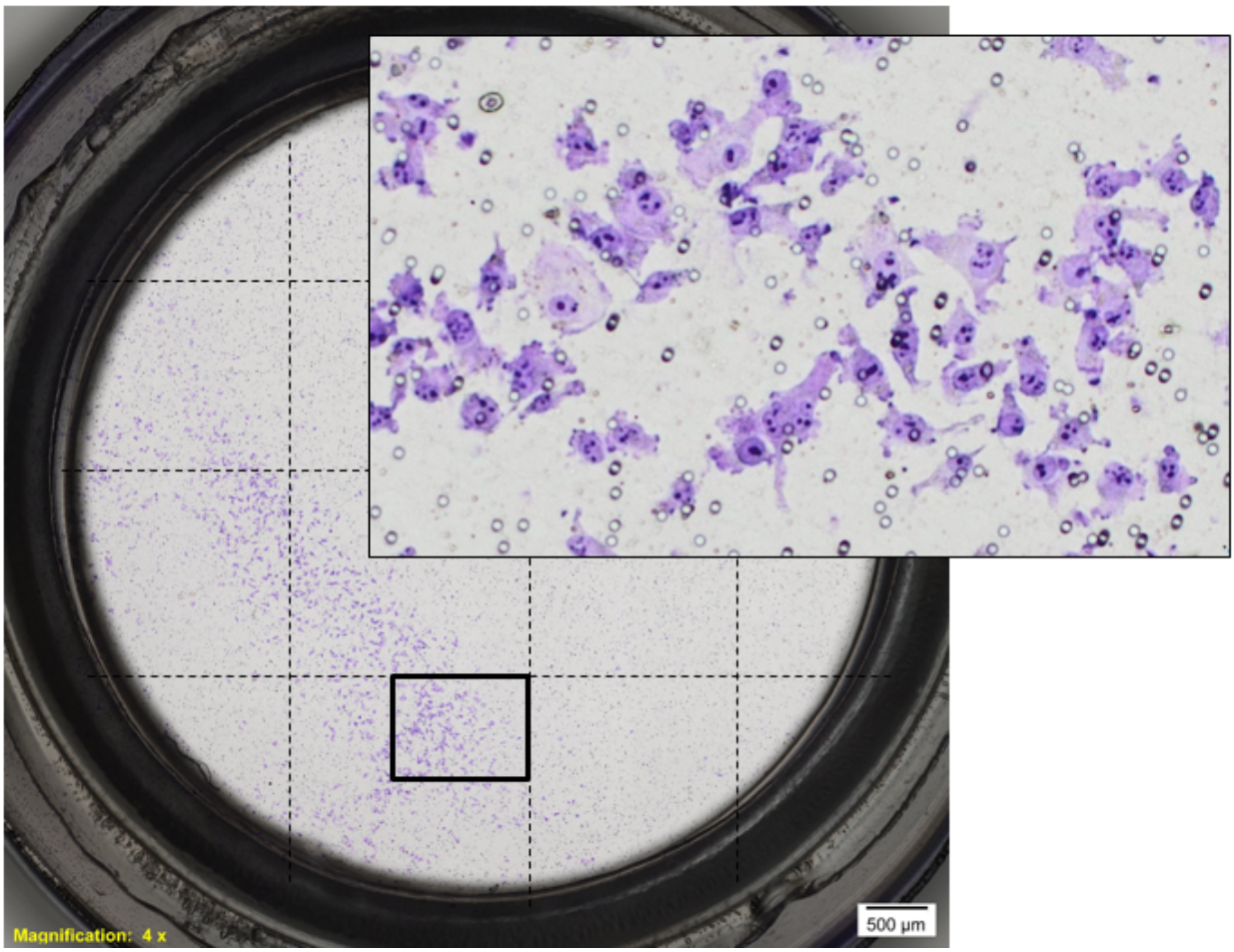


Figure 5.1: Original pictures of tumorous cells grown on a porous membrane are too large to be analysed as one block. Fraction of the complete picture is processed at the time. This sample has been used for the development of the detection algorithm.

As the problem is the detection of elements, detecting their boundaries could be fine solution. Several edge detection filters exists. We tried the direct application of edges detection on an untouched original picture. The results (presented in Subsection 5.1.2) were not conclusive as the cells boundaries are not well defined and pores are too visible.

Before counting the cells, the first stage is therefore to get rid of the background. One detail that differentiates cells from the rest of the image is their colour. It ranges in a specific purple palette. By filtering the image based on its colour, we can eliminate most of the background. Pores, however, might stay as their contours are darker. Compared to cells, they are thin. Combining erosion and dilation on the mask obtained with the color filter

will remove small isolated elements, such as pores.

The second stage is the discrimination of cells even if they might might come into contact each other. We apply the same steps as before but with a more restricted colour range so that only the nucleoli are preserved. There might be several nucleoli in one nucleolus (hence in one cell). The dilation will help to merge the closest elements together so that they will be counted as one. Once we obtain the black and white mask indicating the regions of interest, the last step is to count them.

The details of the resulting protocol are presented in Section 5.2. Each step was tested separately to study the effect of the variation of their parameters. Section 5.6 compare the cells detected using this protocol with the ones detected manually.

5.1.2 Edges Detection

As the problem is the detection of elements, detecting their boundaries could be fine solution. Several edge detection filters exists. We tried the direct application of edges detection on an untouched original picture with the 'Canny' method of openCV.
The method used is `edges = cv2.Canny(img,100,200)`. The first argument is a single-channel 8-bit input image, the two others are two thresholds for the hysteresis procedure. The output is an edge map. More details are available on openCV website [41].

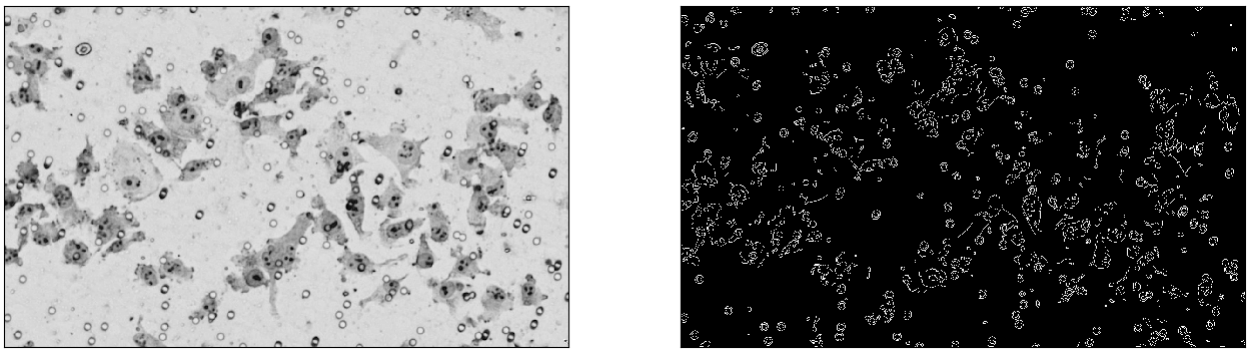


Figure 5.2: Direct application of an edge detection filter (open CV - Canny) on the raw picture. Figure 5.3 shows a close-up.

The Canny Edge Detector can be divided into four steps [41]:

1. A Gaussian filter: smooth the image and remove the noise
2. A procedure close to Sobel filter to find the intensity gradient in the image
3. Suppression of non-maximum pixels
4. Hysteresis using two thresholds.

Figure 5.3: Zoom of the direct application of an edge detection filter. Pores appears clearly (red arrow). The image is too noisy, there is too much information.

The result was not conclusive. There is way too much noise. Indeed, pores are detected as well as any other biological noise (see Figure 5.3).

5.2 Protocol

The protocol used in this thesis is the following:

1. color filter
2. erosion of the mask
3. dilation of the mask (= opening)
4. second filter
5. potential additional MM
6. detection of groups from the resulting mask

We begin with a color filter. Indeed, cells, when observed under a microscope are usually dyed so that they become more recognizable. This means that they adopt a distinctive hue. By filtering the image to keep this range of colour, we can eliminate the background.
This is a very important step because it reduces the search field. The output of this step is a binary mask that is composed of white pixels where the original image meets the criteria.

Despite this first selection that removes a big part of the image, there are still small pieces in the mask that are definitely not cells. Indeed, on the original image, cells stand on a porous membrane. And those pores are dark and tends to be included into the first selection. This is why the mask then undergoes an erosion. This operation aims to delete all elements too small to possibly be a cell. Then, to restore the initial volume of the selection, a dilation is applied. If both are performed with the same structuring element, this can be considered as an opening.

Now that the mask should be free of pores, there is still an other problem to deal with. Cells are likely to be really close, if not superimposed. Distinguishing their exact boundaries is a real challenge but there is one thing that each individual cell possess: a nucleus.
This nucleus adopt a darked hue but still is not contrasted enough. In Figure 5.1, we can see that each of those nucleus has even darker spot in them. We make the hypothesis that those really dark and small dots are nucleoli, condensed DNA. The second hypothesis is that if they are close enough, they are part of one nucleus. The manual identification in Figure 5.4 reinforces those hypothesis.

The second filter aims at finding each nucleolus among the sub-selection. Then, an additional dilation on the resulting mask will merge very close elements so that they represent a single entity, hence a single cell. An other erosion can also be performed beforehand to smooth the result.

After some research, it appeared that a similar protocol was recommended by [8] to eliminate noise.
"After thresholding the image, you'll see small white isolated objects here and there. It may be because of noises in the image or the actual small objects which have the same color as our main object. These unnecessary small white patches can be eliminated by applying morphological opening."

Visualisation of the result

At each stage of the treatment, it is very important that the user is able to visualise the result of the step he has done relatively to the initial image.
This is why a script has been developed to obtain a clear superimposition of each processing. This way, it is easier to notice early if a cell is being excluded due to a wrong parameter or if too many 'noise elements' are still included in the selection. In Figure 5.5a, the area that are part of the selection are highlighted by their HSV color. Cells appear pink and the darker purple becomes green. To get this result, we call `showMask()` with the background and the mask to superimpose. There is also an optional argument: `highlight`. When it is at `False`, the mask will be represented in an uniform red color instead of HSV colors.

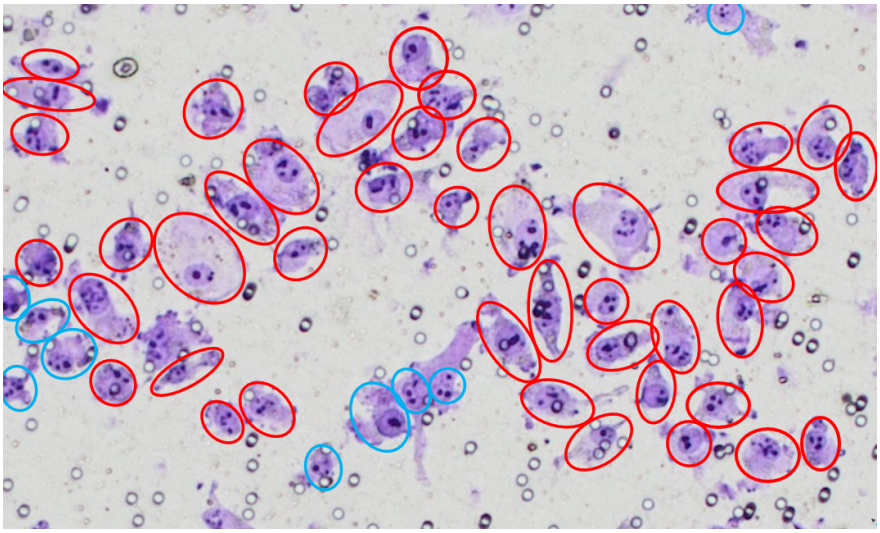


Figure 5.4: Manual identification of cells based on indications of Marie Fourrez (red ones) and the hypothesis that each cell can be identify by its nucleus which is a darker and round element in the cell (blue ones). The manual counting gives 54 cells.

```
def showMask(background, mask, highlight=False):
    ## Now create inverse mask
    mask_inv = cv2.bitwise_not(mask)
    ## create color mask

    if highlight:
        imgHighlight = cv2.cvtColor(background, cv2.COLOR_BGR2HSV)
        colourTag = imgHighlight
    else:
        height, width = mask.shape
        blank_image = np.zeros((height, width, 3), np.uint8)
        blank_image[np.where((blank_image == [0, 0, 0]).all(axis=2))] = [22, 37, 234]
        colourTag = blank_image

    ## superimposition
    ## The two images to add (bitwise)
    img1 = background.copy()
    img2 = cv2.bitwise_and(colourTag, colourTag, mask=mask)
    # cv2.imshow('mask2',img2)
    ## If I want to put logo on top-left corner, I create a ROI
    rows, cols, _channel = img2.shape
    # roi = img1[0:rows, 0:cols ]

    ## Now black-out the area of logo in ROI
    img1_bg = cv2.bitwise_and(img1, img1, mask=mask_inv)
    ## Take only region of logo from logo image.
    img2_fg = cv2.bitwise_and(img2, img2, mask=mask)
    ## Put logo in ROI and modify the main image
    dst = cv2.add(img1_bg, img2_fg)
    img1[0:rows, 0:cols] = dst
    return img1
```

5.3 Filter the color range

The first step is the elimination of the background. There is a number of different elements that are not relevant in the cell recognition.

- The background, which is the lighter
- The pores, very small dark circles
- Organic noise, slightly purples elements that may have detached from cells or being part of the nutritive environment

All elements listed above could be removed without risking to loose a cell in the process. For the sake of simplicity, we consider that they are the only external elements. The microscope edges shall not be present on pictures.
The image is composed of a rather narrow color palette. And cells, due to the dye used, are contained in a purple gradient. By using a classic color filter, it should be possible to exclude some non-relevant elements from the image that shall be analysed.

This is done using the `nucleusIsolation.py` script (see Figure A.4 in appendices). It works as follows.
OpenCV usually captures images and videos in 8-bit, unsigned integer, BGR format. This means that the channels are not in the same order as in the classic RGB colour system. HSV color space is the most suitable color space for color based image segmentation [8]. The original image is therefore loaded and then converted to HSV.

A mask is created using `cv2.inRange(imgHSV, min_purple, max_purple)`[1]. This output is a binary matrix. Similarly to a classic thresholding (but applied to a multi-channel image) the pixels with HSV value in the range are replaced by 1 and the other becomes 0. The mask is more easily modifiable than the original picture and being binary, it is more adapted for mathematical morphology methods (see Figure 5.6. In order to visualise the extend of the selection, it is better to superimpose the result on the original picture. This is shown in Figure 5.5a.

Several color ranges have been tested to find the one which keeps all the identified nucleus without keeping to much of the background. The observations are in the following subsection.

5.3.1 Colour identification

The differences between RGB and HSV colour system are presented in Section 2.1.4. To make sure that cells and the background have a significant color difference and in order to

[1] cv2 is for openCV

	lower limit	upper limit	
1	[130,50,200]	[145,255,255]	nucleolus are excluded from the selection
2	[125,50,0]	[145, 200, 255]	a fraction of nucleolus are excluded from the selection
3	[90,34,0]	[145, 255, 255]	complete nucleus incorporated in the selection
4	[125,50,0]	[145, 255, 255]	complete nucleus incorporated in the selection + more restricted
5	[125,96,68]	[145,255,255]	selection restricted to isolate only nucleus
6	[65,106,0]	[160, 255, 163]	selection restricted to isolate only nucleus

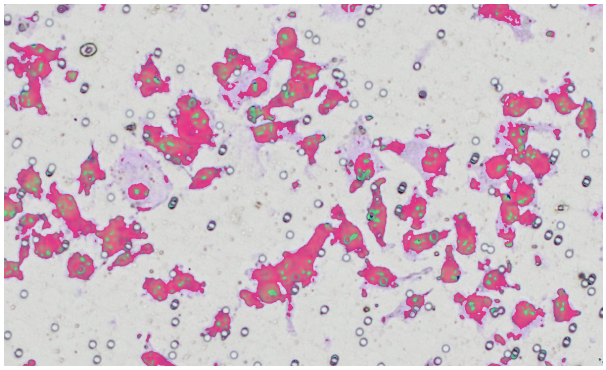
Table 5.1: Observation of different filter ranges on the source image.

successfully segment objects using color based methods, we convert the image in HSV. Then, it is necessary to define the range of color that will be kept.

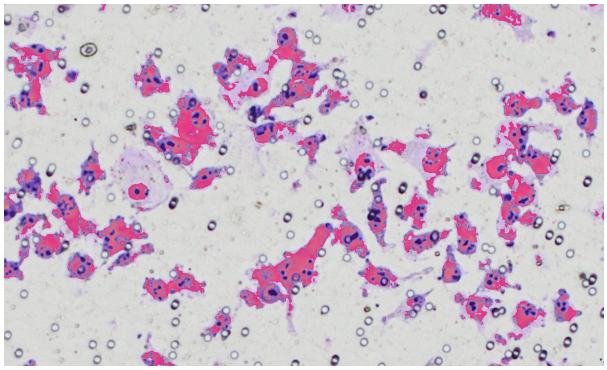
There are a lot of websites where one can find RGB or HSV 'color pickers' with which it is easy to obtain the colour code. However, in OpenCV, value range for 'hue', 'saturation' and 'value' are respectively 0-179, 0-255 and 0-255 [8] instead of the classic H = 0-360, S = 0-100 and V = 0-100, as for Gimp. This means that the converted image will not appear in the same hue than the original. A conversion has to be made.

Different values have been tested in order to find the range which would include all the cells and exclude the maximum 'noise' element at the same time. In Figures 5.5a and 5.5b, we can see the comparison of two results. Surprisingly, the criteria which has the most impact is the *saturation*. But incorrectly calibrated, the *value* can also impact the quality of the selection. Figure 5.5b shows an example where nucleoli end up excluded from the selection due to a too narrow value range. According to [8], the *hue* for purple is supposed to be between 130 and 160 on a range of 179 (see Section 2.1.4). However, with tests at a constant *saturation* and *value*, it appears that the ideal hue was between 90 and 145. This is strongly dependant of the processed picture.

Table 5.1 gathers some significant observations. From there, the filters identified to be the more adequate for samples of Figure 5.1 are the number 4, for the first restriction, and number 6, for the nucleoli isolation.



(a) Filter with range [125,50,0] - [145,255,255]



(b) Filter with range [130,50,200] - [145,255,255]

Figure 5.5: Comparison of two different range of color. On the first one we can see that nuclei and nucleoli (highlighted green in HSV) are still included in the selection. The second one has a narrow value range. The consequence is that the nucleoli are excluded.

5.4 Mathematical morphology

After the first processing, there are still 'noise' elements in the image. Indeed, pores have a dark colour that is included in the color range. This can be resolved by some operations on the binary mask. If pores are selected, they have a really thin shape compared to cells. Mathematical morphology will help to eliminate them from the mask.
From the technical point of view, note that erosion and dilation can be obtained with the script `mathMorpho.py` (see Figure A.5 in appendices). Several *Structuring Element* sizes have been tested. This is described in the following subsections.

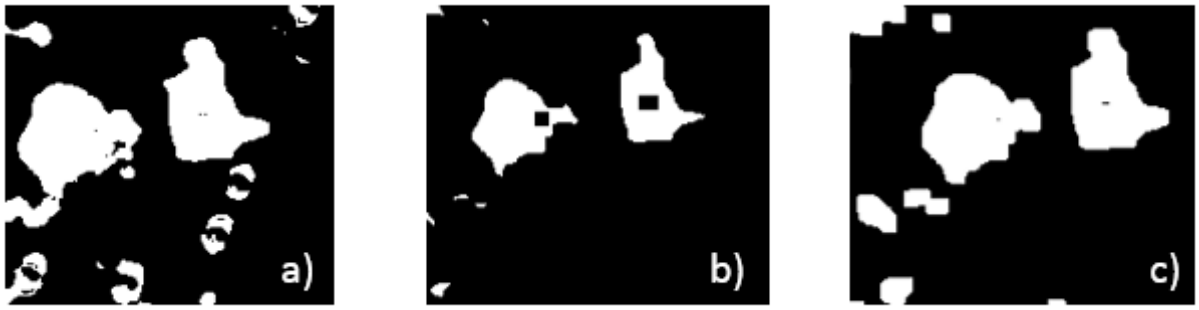


Figure 5.6: Transformation of the binary mask.
a) mask after the colour filter, b) mask after erosion, c) mask after dilation

5.4.1 Erosion

An erosion eliminates the smallest elements from the binary mask. All depends on the size of the structuring element. It shall not be to small or the parasitic element will stay. But a

SE	observations
[5,5] and smaller	Several pores are still part of the selection
[6,6]	Most pores are excluded from selection
[7,7] - [9,9]	Pores are excluded, cells space start to be reduced
[10,10]	The smallest cells start to disappear
from [20,20]	Most cells are excluded from the selection

Table 5.2: Observations for an erosion applied on the mask resulting from a colour filter of range [125,50,0]-[145, 255, 255]

too big SE could start to erase the cell from the selection.

The first intuition was to use round structuring elements to match the cell natural shape. However, this was not possible with openCV. The structuring element (also called kernel in openCV) was therefore defined as a square which is the closest. For the programming, erosion is obtained with the following lines:

```
kernel = np.ones((SE, SE), np.uint8)
mask_eroded = cv2.erode(mask, kernel, iterations=1)
```

The kernel is created using Numpy. It is a square element and SE is here the size of it. This kernel is then used by openCV to erode the mask (binary file) using the 'erode' method. The output has the same dimensions as the input 'mask'.

We compared several square structuring elements on each mask resulting from previously mentioned color filters (see Table 5.1). Table 5.2 contains the results for the 4th filter.

The results depend a lot of the colour filter. There is a tradeoff to make to make sure that all pores are eliminated from the selection and at the same time, that the cells are not excluded from the selection. The best combination observed are a restrictive colour filter (filter 4 from Table 5.1) with a SE of size 9 or a less restrictive filter (filter 3 from Table 5.1) with a SE of size 10. In the first case, it is necessary to perform a dilation before going further in order to regain space around cells in the selection.

5.4.2 Dilation

Dilation is done thereafter to restore selection (fill up the holes) and avoid to separate one cell into several elements. The combination of erosion and dilation is similar to what is called an *opening* except that the structuring element is not necessary the same size. An illustration of the effect of dilation can be found in Figure 5.7. For the programming, dilation is obtained with the following lines:

SE erosion	SE dilation	observations
[9,9]	[4,4] and smaller	Holes in the cell selection are still there
[9,9]	[6,6]	Small holes in the cell selection are still present
[9,9]	[8,8]	All cell selections are filled up
[9,9]	from [17,17]	Separate shapes start to merge and some pores are reintegrated to the selection

Table 5.3: Observations for an erosion followed by dilations applied on the mask resulting from a colour filter of range [125,50,0]-[145, 255, 255]

```
kernel = np.ones((SE, SE), np.uint8)
mask_dilated = cv2.dilate(mask, kernel, iterations=1)
```

The kernel is, once again, a square element of size SE created using Numpy. This kernel is then used by openCV to dilate the mask (binary file) using the 'dilate' method. The output has the same dimensions as the input 'mask'.

Different SE have been tested. Pertinent observations are gathered in Table 5.3. For the first phase where the aim is to get rid of pore while making sure that no cells is excluded, it is important to choose a dilation with a SE of minimum 8 x 8. A larger SE is possible but may not be necessary. It might instead increased the risk of finding undesired elements back into the selection.
For the second stage (see Subsection 5.4.3), where only element left are supposed to be nucleoli, the aim is to merge the closest together. A SE of 12 is suitable. A larger SE might merge nucleoli from different cells together.

5.4.3 Finding nucleoli

In second stage, undesired elements have been eliminated, a new selection based on color can be done. It is conceived to be more restrictive than first stage so that we only keep the nucleoli i.e. dark purple elements present in the selection. Figure 5.8 shows them highlighted in green.

From this second restriction, nucleoli tends to be isolated from each other. However, there might be more than one nucleolus is a simple cell. To avoid counting them as multiple entities, we use dilation one more time. The aim is to fuse very close elements that are more likely to be part of one cell (see Figure 5.8).

5.5 Counting

Now that we have a binary mask that locates the cells nuclei, they can be counted. In binary images, finding contours is way easier. This step can be found in the script `enumeration.py`

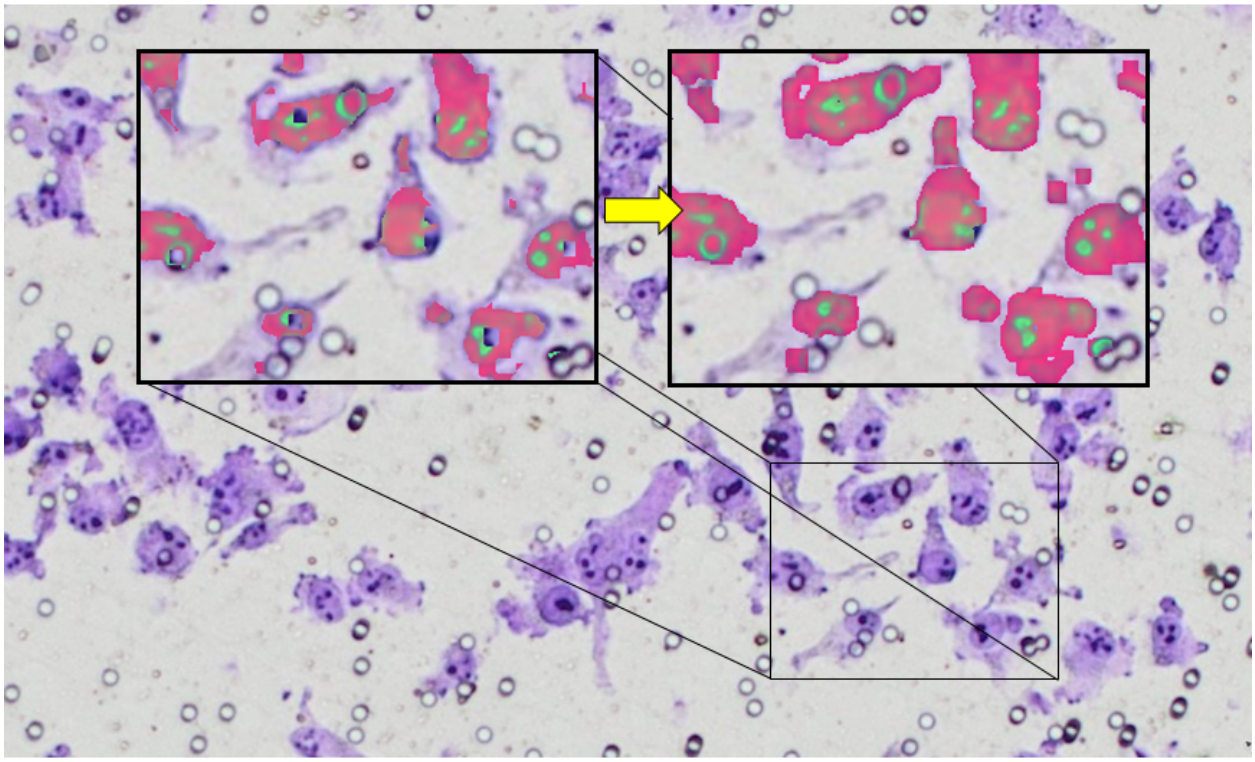


Figure 5.7: Effect of a dilation with a SE size 8 on a previously eroded (SE size 7) selection.

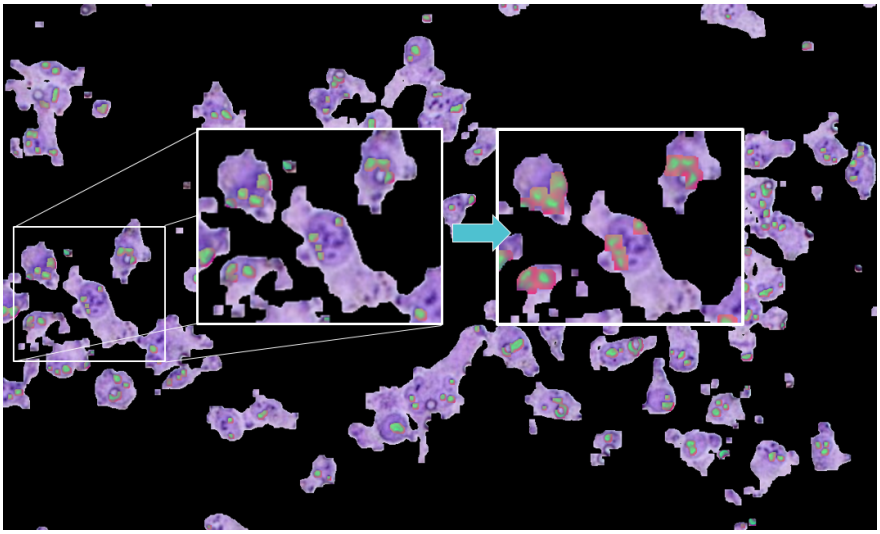


Figure 5.8: Application of a second filter on a restricted picture. The pores are out of the picture, which means that the only dark purple elements left are likely to be nucleoli. Here the filter applied has the following boundaries: [65,106,0] - [160, 255, 163]. Then an erosion with SE of size 4 and a dilation with SE of size 12 have been applied to merge close nucleoli together.

(see Figure A.6inspired from [43]. It contains the following steps.

1. Simplification of the image using conversion to gray scale and threshold.
Not necessary as we already have the mask that is a binary image.

2. Identification of contours is done with the following method.

```
contours, hierarchy = cv2.findContours(img2, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
```

`findContours()` is one of the most frequently used OpenCV algorithms to segment objects that uses Green's theorem and image moments. There are three arguments. The first one is source image, second is contour retrieval mode, third is contour approximation method. The outputs are a Python list of all the contours[2] in the image and their hierarchy. It works best on a pre-processed image. It is often applied after an edge detection filter could do but any binary images can be processed. From there, we can directly draw them on the original image to show what has been detected as a cell using:

```
cv2.drawContours(background,contours,-1,(0,255,0),3)
```

For this method we provide the source image (background), a Python list containing contours. The third argument define if you want to draw an individual contour (then this is the contour index) or draw all contours (then pass -1). The two last arguments are the color and the thickness.

3. For each contour which has an area greater than a given factor ('fidelity criteria'), the contour is counted, converted to a rectangle and added to a new list.

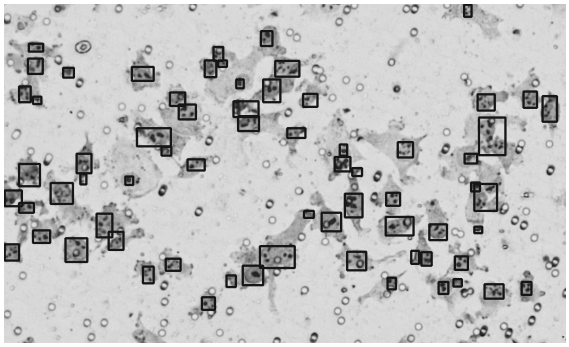
```
br = []
for i in xrange(len(c)):
if h[0][i][3] == ROOT_NODE and cv2.contourArea(c[i]) >= fidelityRange:
totalContours += 1
approx = cv2.approxPolyDP(c[i], 3, True)
br.append(cv2.boundingRect(approx))
```

4. Every rectangle from the list is added to the original image to frame the cells.

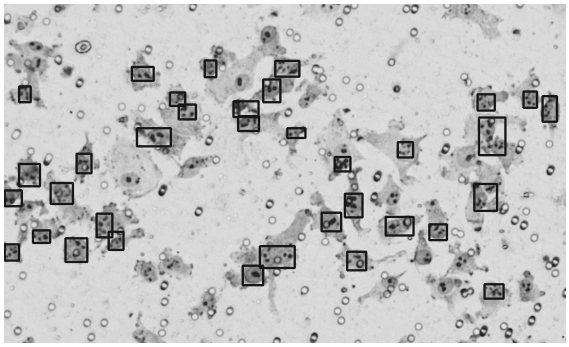
```
for b in br:
cv2.rectangle(imgCopy, (b[0], b[1]), (b[0] + b[2], b[1] + b[3]), (255, 255, 0), 3)
cv2.imshow('image',imgCopy)
cv2.waitKey(0)
print 'Total contours: ', totalContours
```

The fidelity criteria is an additional means to control what will be counted as a cell. Its impact is illustrated in Figures 5.9a and 5.9b. Setting a minimum area prevents some tiny residual noise to be added to the total. The exact boundaries of cells are not important so we can approximate their contour by a rectangle. It will be useful for displaying which cell has been counted.

[2] a Numpy array of (x,y) coordinates of boundary points of the object



(a)



(b)

Figure 5.9: Application of the cell enumeration on the pre-processed masks. In Figure a) (fidelity fixed at 200), there is still some non-cell elements that are counted whereas in Figure b) (fidelity fixed at 600), we can see that there are cells that are not counted.

5.6 Comparison with the expected result

Based on the initial hypothesis, the cells are supposed to be detected as depicted in Figure 5.4. This cell identification does not pretend to be the right one but was necessary in order to compare the results obtained with our approach and the "reality".
The selection in Figure 5.10 has been obtained with the following processing. The images resulting from each steps for this set of parameters are shown in appendices.

1. Filter 4: [125,50,0]-[145,255,255] (Figure A.7)
2. Erosion with SE size 9 (Figure A.8)
3. Dilation with SE size 8 (Figure A.9)
4. Filter 6: [65,106,0]-[160,255,163] (Figure A.10)
5. Erosion with SE size 4 (Figure A.11)
6. Dilation with SE size 10 (Figure A.12)

The final number of cell counted is 55. This is 1.85 % more than the manual identification of Figure 5.4.
This very small difference seems encouraging but it is a little bit misleading. Even if most of the detected elements are indeed cells, there are errors. Unfortunately some cells are missed out (Figure 5.11.a), some are fractioned and counted several times (Figure 5.11.b) and there are still elements that are counted as cells even if they are not. This happens with pores that are under the cells and therefore, are protected from erosion (Figure 5.11.c). In this result, the number of non detected cells is 8 out of 54 manually counted in Figure 5.4. This means a little more than a 14 % false negative. This is more than the maximum tolerance of 5 %. We should however keep in mind that in the end, what matters the most is the ratio between the pre and post-treatment cells. And if this error percentage is constant, it will

not influence the final ratio.

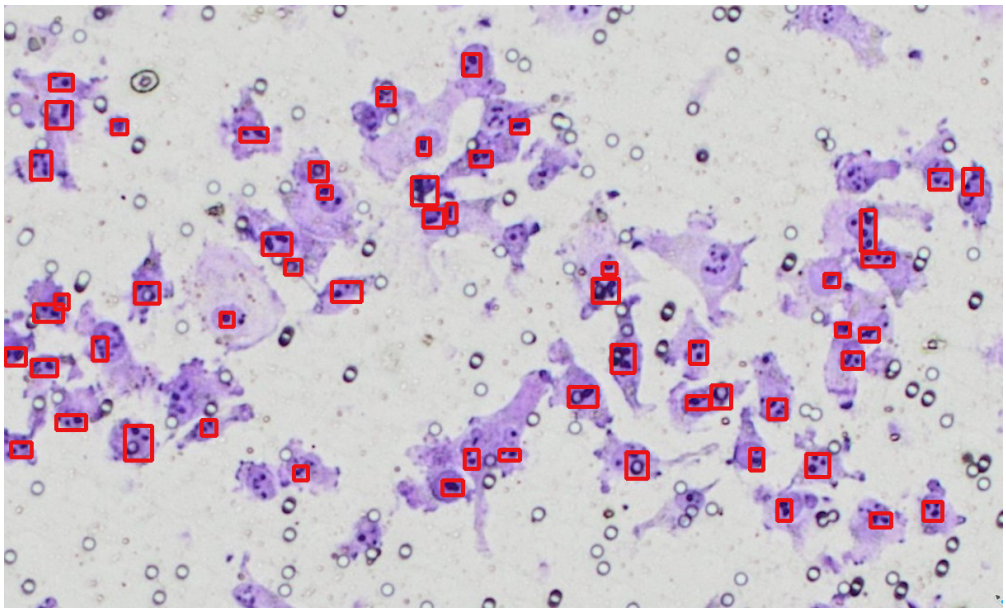


Figure 5.10: Cell count obtained with the optimized processing: filter 4, erosion of 9 and dilation of 8, followed by filter 6, erosion of 4 and dilation of 10. The number of cells reported is 55.

To address the accuracy problem, we added a functionality that enables a manual correction of the result by the user. It is explained as part of the user interface development in Chapter 6.

5.7 Conclusion

In this chapter, we tested the chosen protocol. First a filter that restrict the selection to potential cells. An erosion is then applied to remove noise elements, including pores and a dilation helps to restore the selection. This treatment is applied a second time with a more restrictive colour filter. This time, the aim is to isolate nucleoli. The mathematical morphology is then used to merge together the closest element to avoid them to be counted as several cells. The cells are then counted using the resulting mask.

An optimized algorithm has been proposed. On the output, the number of pores excluded is close to 99 %. However, there are still errors like cells missed out or counted as several ones. This can however be adjusted using the different parameters.

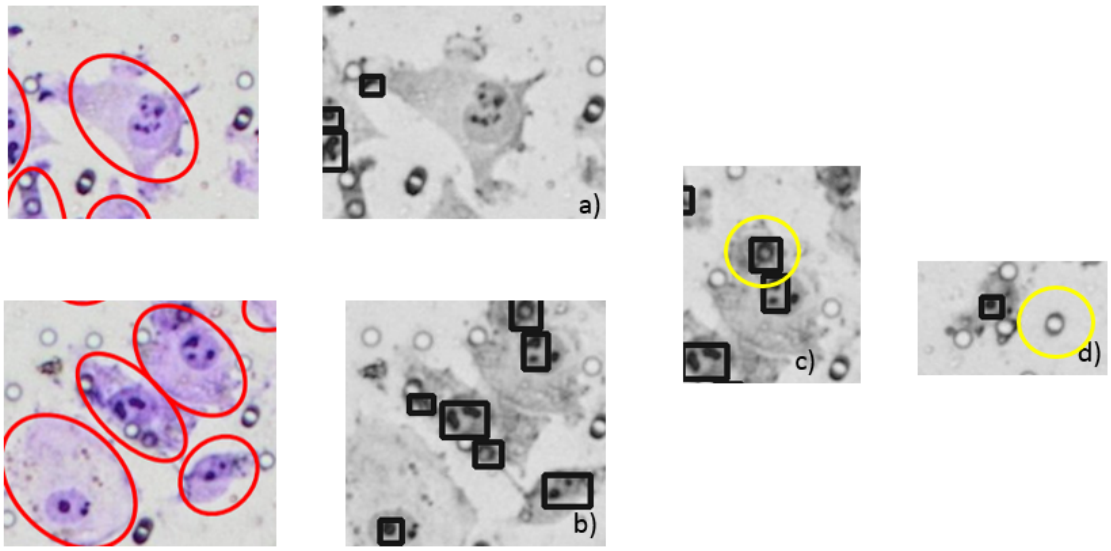


Figure 5.11: Illustration of error that can happen.
a) a cell that remains unselected, b) cell fractioned, c) pore under a cell is being counted, d) pores outside cells are ignored

Chapter 6

Development: Graphic user interface

The second objective of this thesis is to propose a way of using the previously developed algorithm. We will go through requirements of a graphical user interface (GUI) and then compare several options.
Choosing the appropriate support for the cell counting application is not an evident task. As it is not possible to test all possible framework exhaustively, we chose to investigate three solution: a raw Python script, a Python framework for GUI, and a web based application.

6.1 Requirements of a GUI

In order to offer to biologists the opportunity to manipulate their images and use separately the different tools, the interface must meet several essential requirements:

- Load and display the original image
- Apply filters on the original image and visualise the result
- Compare the original and the result
- Enter parameters to tune processing
- Obtain the number of cells detected
- Apply a default processing
- Should be applicable to at least subsection of the complete picture ("zoom")

From there, an exploration sketch has been made (see Figure 6.1). There are also requirements that are not necessary to a first functional version but would greatly improve the user experience. They are called secondary requirements:

- Add as many filters as desired

- Decide the order of filters
- Sample colours directly on the picture
- Manually cancel or add cell on the result of detection.
- Memorize each step and allow 'undo' action
- Automatic segmentation of the full picture and 'intelligent processing'

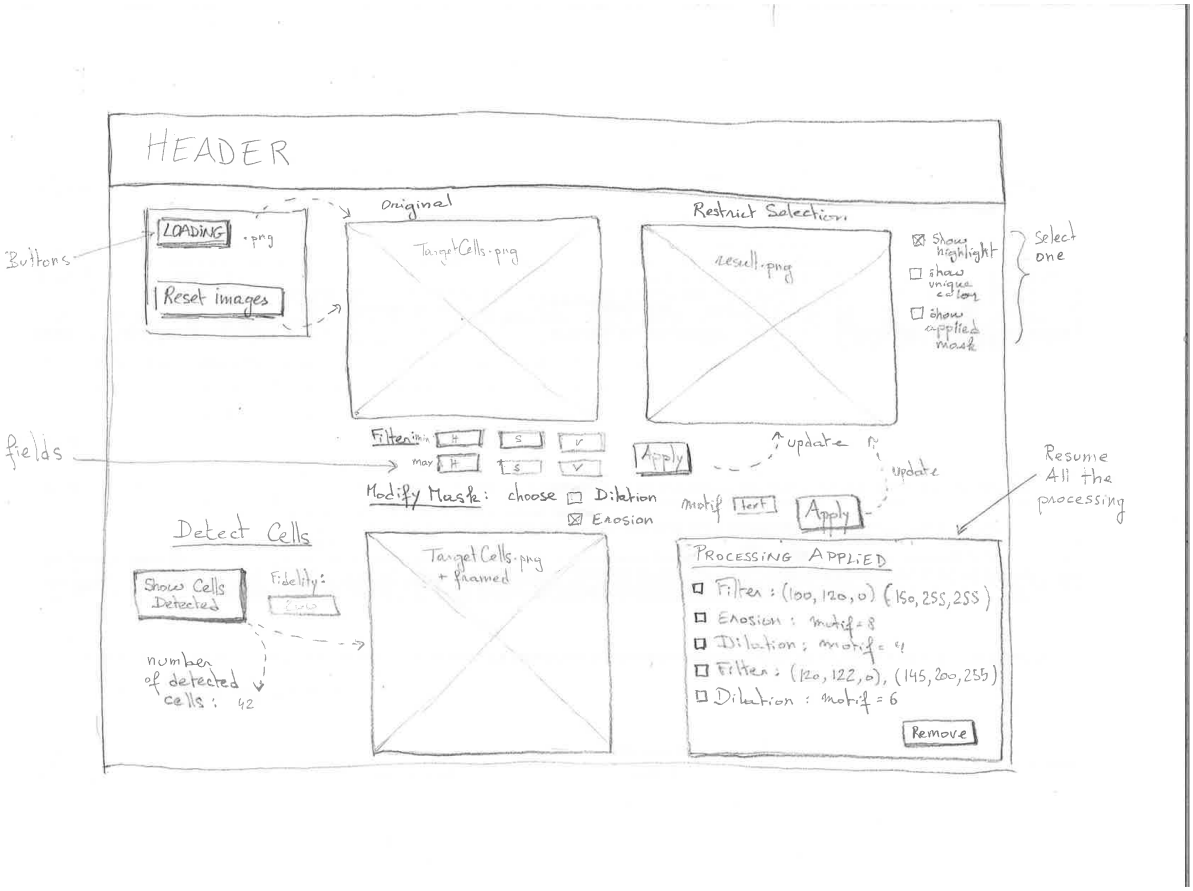


Figure 6.1: First exploration sketch. The Cell Counting application should display the original image and the result of processing. From that, the image with detected cells can be displayed. A resume of all the steps could be also available.

6.2 Choice of framework

Aside from the solution where the user runs directly a Python script with several parameters, we need to build an actual graphic interface. Many solutions are available. The choice of a framework is a crucial decision.

Ideally, the framework should allow the following characteristics:

- **dynamic interface**: once a processing has been applied on a picture, the result shall be directly displayed.
- **easily accessible**: the solution shall be accessible to the general public. Without additional installation.
- **applicable to most computer configurations**: Linux, Mac or Windows
- **compatible with Python**: the image processing is done in Python. The communication shall be possible between front-end and back-end.

Aside from that, there are also functionality that are required:

- **loading an image**
- **applying processing to an image**
- **providing parameters as input for the processing**
- **displaying the original image and the result of processing**

Frameworks offer a structure for application development. They cut the development time by automating the implementation of common solutions. Python offers a huge number of options to provide graphic interfaces. The 'python wiki' lists not less than 36 cross platform frameworks, 7 platform-specific frameworks and 26 GUI design tools and IDE. Platform-specific frameworks, like Ocean or MacPython, were directly excluded for both usability and development reasons. It was not possible to compare all of the remaining possibilities, therefore, based on characteristics available online and existing comparative evaluations, one solution has been selected and compared to a web-based REST solution. A web-based application is any program that is accessed over a network connection using HTTP, rather than existing within a device's memory [44]. It has the advantage of being easily accessible. Indeed web-based applications often run inside a web browser.

6.2.1 Python GUI framework

The selected Python GUI framework is appJar. Designed to run on as many versions of Python as possible, appJar is intended to be simple. Unlike PyQt, it is free. Few lines are enough to start a basic interface. Moreover, tutorials are available at `http://appjar.info/`.

6.2.2 Web based application

A web based interface constitutes a good way to set an easily accessible application. This is a popular solution which allows a lot of customisation of the graphic interface. However, this is also a much more time consuming solution regarding developments.

Among all of those frameworks, a lot of them are quite limited in terms of functionality. One essential feature is a good image manipulation toolbox as the aim of the software is to analyse and dynamically manipulate images.
The size and the complexity of the project is another non negligible aspect. As the application can stay small and simple, and as the time to develop was limited to months, a micro-framework could be considered. It is suitable for a "proof-of-concept" solution.

For this master thesis, the development platform selected for the front end is Angular. As for the background, preference was given to a python server as the image processing was entirely done in this language. Django, Pyramid or TurboGears are called Full-Stack frameworks. They are suitable when developing a large system packed with features and requirements but bring a set of limitations.
The most famous micro-frameworks are Flask, Bottle and CherryPy. If the last one is open-source, Flask present many qualities to help building a solid web application and adapt to developer's need. Bottle implements everything in a single source file. It limits dependencies and is more appropriate for prototyping.
It is Flask that has been selected.

Angular

Angular is a web app framework. It has been developed by Google and has the advantage of being open-source (over two thousands contributors at the time of Angular 2). It is what we can call a "client side" framework. It will be used to dynamically handle the graphic interface in a way that is independent from the functionalities and the server language.

Angular-enriched HTML and JavaScript offers features that make it easier to implement an application with complex requirements. It is especially adapted for applications that involve collecting data from forms and process those data.
Even if it is really powerful, Angular is not the most simple or easy-to-use framework. It takes some time to get a grasp on it but once this is done, developers can obtain robust and clean results in a small amount of code [45].

Flask

Flask is a python microframework. Like Django, it is used to develop solid web applications but is focused on providing a functional and concise core. If Django is more adapted to develop a long-term end product, Flask is considered much more lightweight and experience focused [46]. It is suitable for working application from the ground up in a short amount of time which is the description of the application developed in this thesis.

A very complete comparison of Python and Django made by Shamikh Hossain, CS Research Assistant at Duke University, and Michael Yousrie, Python/PHP Full Stack Web Developer [46] sums up the main arguments in favour of Flask.:

- written in python
- comes with the very basic minimum required elements to get a web app up and running as fast as possible
- requires less configuration than Django
- does not force a specific structure

Flask is available under the BSD license [1]. It does not depend on the front end technology and provide useful out-of-the-box feature like:

- built-in development server and a fast debugger
- HTTP request handling
- RESTful request dispatching
- integrated support for unit testing

6.3 Implementation

6.3.1 Python with no GUI

The first and the simplest solution is to have no graphical user interface.
The functions are gathered into a package, `CellsCountPack`. The user has to run a method to process its image.
The parameters are the source image path, the limits for the color filter and if necessary, the size of the structuring element for erosion and dilation. The following methods are offered:
`restrictSelection(imgSourcePath,min color, max color, SE erosion, SE dilation)`
`restrictAndCount((imgSourcePath,min color, max color, SE erosion, SE dilation, fidelity)`

[1] family of permissive free software licenses, imposing minimal restrictions on the use and redistribution of covered software [47]

The first one returns a restricted version of the picture while the second uses the computed mask to count the cells. It displays the number of cells and creates an image with the framed cells. There is also a version with 'viewer' which allow the user to find the lower and upper colour limits by trying dynamically on the image. A screen shot can be found in Figure 6.2.

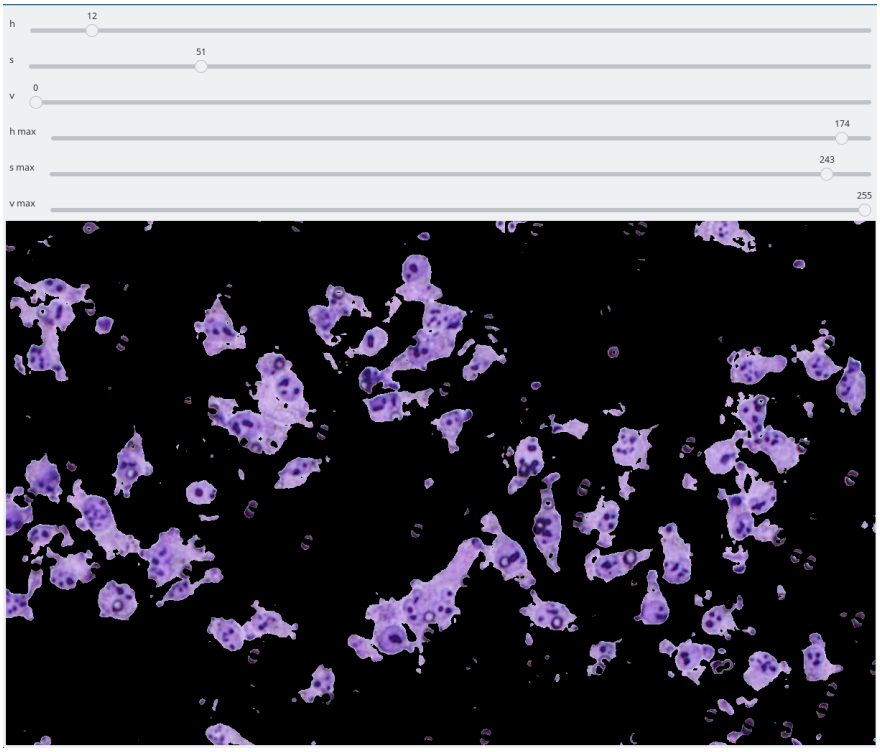


Figure 6.2: OpendCV - python viewer. It allows the user to directly visualise the results of a color range in the color filter.

As the results of the algorithm is not always perfect, an additional tools has been implemented. When using the `restricAndCount()` method, the user will have the choice to correct the detected cells. He can add or delete frames like the ones on Figure 5.10.

The advantages of a raw Python script are:

- It is quickly implemented
- The code is accessible meaning that it can be adapted
- More flexible: the user has a whole library that he can use as he wants.
- Other Python tools can easily be integrated. openCV display feature can be used without restriction.

The disadvantages of a raw Python script are:

- It is less intuitive

- It requires coding skills in order to use it right
- Code accessible means that it could be altered
- It requires a solid documentation

`CellsCountMathods.py` must be parametrized directly in those lines:

```
##### HOW TO USE THIS SCRIPT
### Enter the path to your image in the image_path variable
### Enter the parameters for the first restriction:
## color range limits, SE for erosion, SE for dilation
### Enter the parameters for the second restriction and counting:
## color range limits, SE for erosion, SE for dilation (and fidelity)
### You can configure the name for the saved result (imwrite)
### run the script. Pictures appears for the different steps.
## Click any key to continue.
### when the detection screen appear, you can rectify the detected cells.
## Draw a frame around the cells that you want to add + click 's' to save them
## Click on the upper left corner of frames you want to delete + click 'd'
## When you're happy with the result, press 'escape'

image_path = directory + '/TargetCells.png'
low_limit1 = [125,50,0]
up_limit1 = [145,255,255]
SE_erosion1 = 5
SE_dilation1 = 8
low_limit2 = [65,106,0]
up_limit2 = [160,255,163]
SE_erosion2 = 2
SE_dilation2 = 12
fidelity_factor = 200

if os.path.exists(image_path) == False:
    print "PATH DOES NOT EXIST"
orig_image = cv2.imread(image_path)
mask = restrictSelection(orig_image,low_limit1, up_limit1,SE_erosion1,SE_dilation1)
res = cv2.bitwise_and(orig_image, orig_image, mask=mask)
#restrictAndCount(directory + '/F6_E10_D6.png',[65,106,0], [160,255,163],4,12,210,orig)
cell_detection, cell_number = restrictAndCount(res,low_limit2, up_limit2,SE_erosion2,
SE_dilation2,fidelity_factor,orig_image)
cv2.imwrite('F1E5D8_F6E2D12_200.png', cell_detection)
```

6.3.2 Python with appJar

A first attempt has been done using appJar as a graphical user interface. appJar provides quick results in minimum time. Useful methods are already present as for the image uploader that is available with the `openBox(imag_ label, default_ location, extension_ tuple)` method (see Figure 6.4).

However, it also suffers from its simplification. The framework was developed for educational purpose and it is meant to hide away the complexity. In such a way that we end up quickly stuck. Images can be scaled up or down (`zoomImage("img", -2)` in Figure 6.3) but their exact relative position is not customisable.

```
app = gui("Login")
app.setResizable(canResize=True)

app.addLabel("lab1","Loading Window")
app.setLabelBg("lab1", "green")
app.setLabelFg("lab1", "white")
app.setFont(16)

app.addLabel("l1", "Load your cell image")
app.addButton("Open", openImg)

app.addImage("img", "TargetCells.png")
app.zoomImage("img", -2)

app.addButton("Exit", exit)

app.go()
```

Figure 6.3: It takes only few lines to create a window using appJar. In this extract, we add a title, a label, a button to open a picture and a space to place that picture.

```
def exit():
    app.stop()

def openImg():
    global file
    file = app.openBox("Images", "/home/arianelit/Downloads/", [('images', '*.gif'),
    ('images', '*.png'), ('images', '*.jpg'),
    ('images', '*.jpeg')])
    app.setImage("img", file)
```

Figure 6.4: Definition of buttons' functionalities. openImg is the uploader. We use openBox() to select the find the file.

The advantage of a python GUI is that at the end, only one language is used and it is less likely to have communication problems between front-end and back-end. An attempt has been made to directly include an openCV window into the interface but it was not really stable.
Figure 6.5 shows an example of appJar layout. Apart from colours, the different elements are not customizable.

The advantages of appJar are:

- quick handling: appJar is simple and intuitive
- code in python: this mean that it can directly communicate with the image processing code
- openCV windows can be lauched from the gui
- functional

The disadvantages of appJar are:

- poor image management: appJar does not offer many option for image display
- very slow for any other extension than .giff and unstable
- the layout is not much customizable due to its educational purpose

6.3.3 Flask and Angular

The solution of an Angular front-end with a Python server is the one that has been the most explored.
Angular has a steep learning curve. This means that even for what seemed basic features, a lot of researches were required. Issues encountered and their resolution are presented in the following sections.
Thanks to time and research, it was possible to create a functional prototype. It allows to load an image and successively apply color filters, erosions and dilations on it. When the processing is fine to the user, he can click on a last button to receive the original picture with cells framed and their estimated number.
Very little time was left for the aesthetic aspects. However, angular works with HTML, the standard mark-up language for creating web pages. HTML is well documented. The customization of the layout offers a lot of possibilities.

In the following subsection, key barriers to the development of an Angular front-end are explained as well as the solutions proposed to address them.

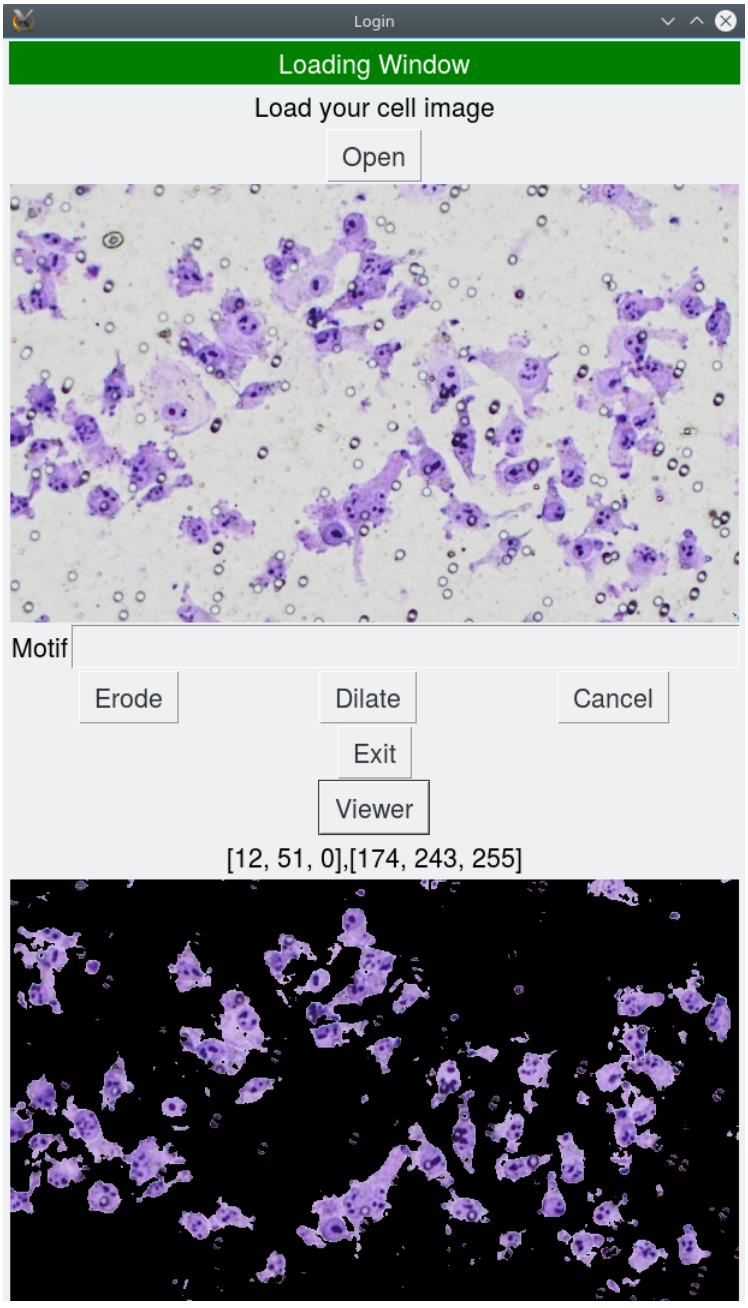


Figure 6.5: First result of an appJar interface. Application of a color filter. The design are poor and the elements disposition is not very flexible.

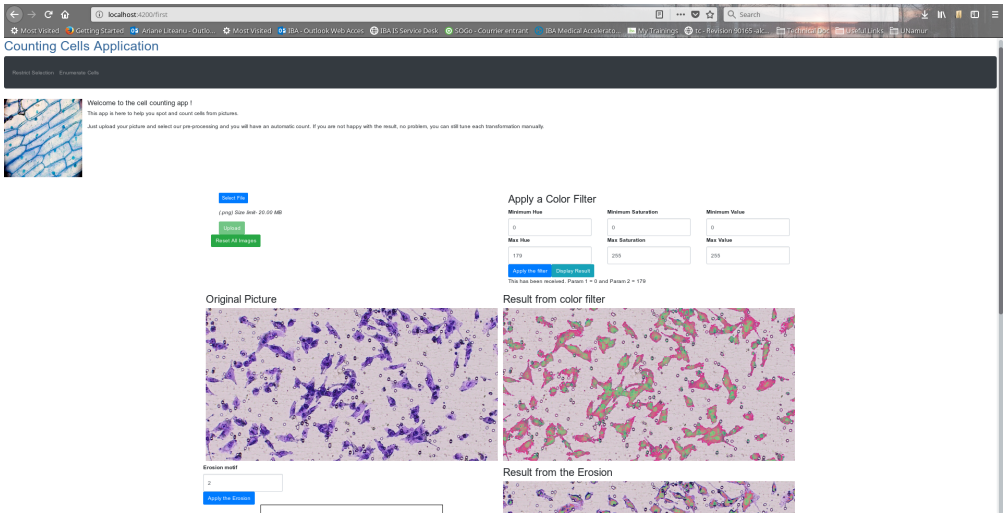


Figure 6.6: Extract from one screen of the application with a Angular front-end. There is an upload section. Colour filter is on the up right and the different results are displayed below.

Load an image

The first crucial point was to be able to load the image which will be processed. Surprisingly, unlike appJar which has a one-line-code solution, Angular has no ready-to-use solution for such a thing. There is no embedded solutions but the angular community offers several possibility to be able to upload files. Some are difficult to apply to a Flask-Angular combination. Since a basic file uploader was enough, the solution selected is `angular-file-uploader` [48] which is a module from *kzrfaisal.github.io* last updated in July 2018. It is compatible with Angular 2/4/5 and 6 and can be installed using `'npm i angular-file-uploader'`[2]. Compared to the other solutions, it is easy to install with only a little configuration. This module will upload the image to an api url. The server side must therefore be configured to receive the image and save it. On the Flask side, an @app.route('/uploader', methods=['GET', 'POST']) is created with a function $upload\_files()$. The file is retrieved using 'request' and saved to a defined folder.
It is important to note that it is still the responsibility of the user to make sure that the file does not have a reserved name. At this stage, it is also possible to refuse a file if it is not one of the expected file extensions. But since this can be handled on the Angular side, this is not mandatory.

Display Images

Angular provides ways to display images. Usually, it is recommended to keep static resources in the Angular folder so that it would be easier to package it later for deployment. However, the cellCounting application needs to process images and then display the result. This means that the display needs to be dynamically updated with new images. If we want to avoid

[2]with Angular CLI

overwriting the images, different names might be used.

One solution to that problem was to define the path to image as a concatenation of strings: a root path and the result name. This way, each time the result name is updated, the result emplacement points to another image.

Local storage

Most of the storage is done through the server which itself saves the images locally on the computer. However, this is not enough to guarantee a smooth experience to the user. Indeed, Angular loses its local variable at each reload (which can happen often) and the user needs to trigger a request to get the information (like the image path) back and make that Angular can display it.

To fix that problem, we therefore need to add a kind of "long-term memory" for the front-end. So that it can recall the recent information at least the time of a session.

One solution is a "LocalStorage". The module `angular-webstorage-service` is an Angular module that gives access to the browsers local storage [49]. By using its feature, it is possible to store information, as the last image name received, into the session. This way, even if the page reloads, it can display the right image. If the session ends (page closed), the information is cleared.

Flexibility

In the conception of the GUI, flexibility was also considered. This means that it should be possible to come back and cancel an action to rectify the result. This means that the simple solution of overwriting a single 'result file' was not satisfactory. To allow a roll back, each step needs to be saved with a different name.

The advantages of an Angular front-end are:

- Separation of concerns back-end front-end
- Maintainability
- Component-based architecture that provides a higher quality of code
- Cleaner code (compared to javaScript alone)
- Great possibilities of layout customization
- Run on browsers
- Higher scalability

The disadvantages of an Angular front-end are:

- No dynamic processing: the processing of an image happens in the server. The user cannot interact directly with the image. i.e. using openCV display
- Steep learning curve: it takes time and research to be able to obtain a functional prototype.
- Verbose and complex
- Not the best for a short term "proof-of-concept": Angular is a strong framework with a lot of features but it is too much for a "proof-of-concept"

6.4 Conclusion

Choosing the appropriate support for the cell counting application is not an evident task. As it is not possible to test all possible framework exhaustively, we chose to investigate three solutions. The first one is to gather the python methods into one package without graphical interface. This is obviously the easiest option for the developer. It is possible to exploit the openCV display options directly. However, the final user must be comfortable with Python. In a second time, we tested a python GUI, appJar. Due to a rush towards implementation, the choice for this framework was probably not the most adapted. Nevertheless, we were looking for a fast result prototype and appJar had the advantage of being really intuitive.
The last is a web-based application. We chose to ally an Angular front-end with a Flask back-end to stay close to the Python image processing methods previously developed. It appeared that even if Angular is promising it is more suited for large and complex applications. It is not the simplest or smallest JavaScript framework. Thanks to time and research, it was nevertheless possible to create a functional prototype. Very little time was left for the aesthetic aspects. However, since it uses HTML, a well documented language, it should not be problematic to customize it.

As said earlier, none of the proposed solution is a perfect match. The Angular-Flask combination was probably the most promising but is not very appropriate for a small application. The solution was kept because of the time already spent on the project but it would have been interesting to try an easier solution. The result is that the application is not "clean", closer to a prototype than a deployable solution.
Angular is often said to be too opinionated and too rigid. The article [45] explains that Angular works best for applications being built by medium or large teams and suggest a simple library like jQuery. In the domain of front-end web framework, there is an other famous framework. REACT is often mentioned as Angular main alternative. While Angular has been developed by Google, REACT has been produced by Facebook. Its strong argument is the freedom and simplicity it offers while keeping most Angular features. A third option, worth to mention, is Vue. It is often seen as an outsider but clearly has nothing to envy to the two others. Lighter than REACT, it is probably the best option for a small project with a limited amount of time for development.

Assuming that we keep a web interface, possible measures of improvement.

- Integrate the dynamic selection windows to allow the user to see directly the effect of a parameter
- Allow to cancel a step or correct it
- Including a 'save' option
- Include the correction of the final result by the user
- Handle complete images and add an 'intelligent', automatic segmentation of pictures that are too large.

Chapter 7

Conclusion

What does it takes to build a functional tool to count cells? This is the question that we tried to answer.
First, knowing your support: we limited our researches to pictures of cells. Before investigating their characteristics and the way to detected them, we recalled what an image is and in that case, it shall be raster images. They have properties like a definition, a bit-depth of a palette and this is by playing with those properties that it will be possible to extract information from the picture. Most of the time, we are interested in the value in the pixel. If we can easily recognize an apple on an image, a computer has trouble to find a meaning to that group of pixel. It can tell their value, the contrast between them, the colour it corresponds to but in therm of feature identification, a computer needs image processing algorithms. In the literature, it is referenced as 'computer vision'.

There are many different ways to transform an image into an object more meaningful and easier to analyse. Most of the time, it includes looking at the content of pixels and gather them with others with a similar nature. Thresholding is quite radical and often used in combination with other techniques. Some exploit already existing information: labels for the classifiers, training sets for machine learning or atlas-based approaches. Region growing, clustering and deformable models will require a starting points, a manual intervention.
A raster image originating from a real element is rarely perfect and smooth. This is the main obstacles of those algorithms. This is where mathematical morphology take action. Compared to the previously described ones, it is a fairly simple techniques which exploit a pre-processing with a binary outcome. By using erosion and dilation operators, the binary image will be smoothed. It is kind of a blind method since it does not care about the initial content of the picture but only the binary shape obtained from a pre-processing.

The protocol that is studied here is a combination of a color filter, mathematical morphology and a contour detection filter. In order to refine the result, the first two are repeated twice. Each time the color filter restrict the selection to a certain element and mathematical

morphology removes noises. Several parameters have been tested. We proposed a combination that produce good results. However, there are still some errors in the detection: some cells are counted twice and others (about 7 %) are missed out.

Three interface solutions have been explored. A raw python script, a simple python GUI (apJar) and a web application with python server (combination of angular and Flask). None of the three have demonstrated a perfect match however, they have noticeable advantages. appJar provided quick results but was also very limited and did not really match with openCV display. The raw script allow to exploit all the possibilities of openCV and a lot of control on the image processing steps but it is also very little user friendly. It requires an expansive documentation and knowledges in python to use it correctly. As for angular, it is known to be a steep-learning curve framework that can provide strong results with a clean code. It is used to create efficient user interfaces. The disadvantage is that it does not permit to use directly openCV display functions and it is not straight to obtain a similar result in javaScript.

Bibliography

[1] Collins English Dictionary. Dictionary.com - picture. http://www.dictionary.com/browse/picture?s=t. 2017-11-19.

[2] Computer Hope. Picture. https://www.computerhope.com/jargon/p/picture.htm, April 2017. 2017-11-12.

[3] Janssens G., Lee J.A., and Macq B. Lgbio2050 - medical imaging course, 2014.

[4] Imedias. Les images vectorielles et les images matricielles. http://www.imedias.pro/cours-en-ligne/graphisme-design/definition-resolution-taille-image/les-images-vectorielles-matricielles/, 2011. 2018-07-23.

[5] Patrick Finot. La différence entre une image bitmap et une image vectorielle. http://www.informatique-enseignant.com/image-bitmap-ou-vectorielle/, November 2013. 2018-07-23.

[6] GomezGraphics. Raster vs. vector. https://vector-conversions.com/vectorizing/raster_vs_vector.html, June 2018. 2018-07-23.

[7] Dzung L Pham, Chenyang Xu, and Jerry L Prince. Current Methods In Medical Image Segmentation. *Annual review of biomedical engineering*, 2(1):315–337, 2000.

[8] Shermal Fernando. Color detection and object tracking. https://www.opencv-srf.com/2010/09/object-detection-using-color-seperation.html, June 2017. 2018-07-05.

[9] Wikipedia. Image resolution. https://en.wikipedia.org/wiki/Image_resolution, November 2017. 2017-11-19.

[10] Stuart Grais. Bit depth. http://facweb.cs.depaul.edu/sgrais/bit_depth.htm, Jully 2017. 2018-07-25.

[11] Sunita Roy and Samir K Bandyophadyay. Face Detection Using A Hybrid Approach That Combines HSV And RGB. *International Journal of Computer Science and Mobile Computing*, 2(3):127–136, 2013.

[12] Darrin Cardani. Adventures in hsv space. *Laboratorio de Robótica, Instituto Tecnológico Autónomo de México*, 2001.

[13] NIH. Genetics home references - what is a cell. `https://ghr.nlm.nih.gov/primer/basics/cell`, August 2018. 2018-08-08.

[14] Eric P Widmaier, Hershel Raff, and Kevin T Strang. *Digital Filters*. Chenelière éducation, 2013.

[15] Don W. Fawcett. Pancreatic acinar cell. cil. dataset cil:10974. `https://doi.org/doi:10.7295/W9CIL10974`, 2011. 2018-01-08.

[16] Alexandre Cunha. Biological image processing with matlab. `http://www.cacr.caltech.edu/~cunha/bi199/three.html`, 2016. 2018-01-08.

[17] National Instruments. Thresholding. `https://zone.ni.com/reference/en-XX/help/370281AC-01/nivisionconcepts/thresholding/`. 2017-07-30.

[18] Chunming Li, Rui Huang, Zhaohua Ding, Chris Gatenby, Dimitris N Metaxas, John C Gore, et al. A Level Set Method For Image Segmentation In The Presence Of Intensity Inhomogeneities With Application To MRI. *IEEE Transactions on Image Processing*, 20(7):2007, 2011.

[19] IN Manousakas, PE Undrill, GG Cameron, and TW Redpath. Split-and-merge Segmentation Of Magnetic Resonance Medical Images: Performance Evaluation And Extension To Three Dimensions. *Computers and Biomedical Research*, 31(6):393–412, 1998.

[20] Michel Verleysen. Lgbio2020 - bio-instrumentation course, 2014.

[21] Zaïane Osmar R. Principles of knowledge discovery in databases - chapter 8: Data clustering. `http://www.cs.ualberta.ca/~zaiane/courses/cmput690/slides/Chapter8/index.html`, 1999. 2018-06-28.

[22] Guy Barrett Coleman and Harry C Andrews. Image Segmentation By Clustering. *Proceedings of the IEEE*, 67(5):773–785, 1979.

[23] Alberto F Goldszal, Christos Davatzikos, Dzung L Pham, Michelle XH Yan, R Nick Bryan, and Susan M Resnick. An Image-processing System For Qualitative And Quantitative Volumetric Analysis Of Brain Images. *Journal of computer assisted tomography*, 22(5):827–837, 1998.

[24] Nameirakpam Dhanachandra, Khumanthem Manglem, and Yambem Jina Chanu. Image Segmentation Using K-means Clustering Algorithm And Subtractive Clustering Algorithm. *Procedia Computer Science*, 54:764–771, 2015.

[25] Lilla Zöllei, Martha Shenton, William Wells, and Kilian Pohl. The impact of atlas formation methods on atlas-guided brain segmentation. In *Proceedings of Medical image computing and computer-assisted intervention: MICCAI International Conference on*

Medical Image Computing and Computer-Assisted Intervention, pages 39–46. Citeseer, 2007.

[26] Yannis Chaouche. Initiez-vous au machine learning. `https://openclassrooms.com/courses/4011851-initiez-vous-au-machine-learning/4011858-quest-ce-que-le-machine-learning`, May 2018. 2018-06-30.

[27] Carsten Steger, Markus Ulrich, and Christian Wiedemann. *Machine Vision Algorithms And Applications*. John Wiley & Sons, 2018.

[28] Kevin Rowe. How search engines use machine learning. `https://www.searchenginejournal.com/how-search-engines-use-machine-learning/224451/`, February 2018. 2018-06-30.

[29] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, And Machine Vision*. Cengage Learning, 2014.

[30] Jean Serra and Pierre Soille. *Mathematical Morphology And Its Applications To Image Processing*, volume 2. Springer Science & Business Media, 2012.

[31] Robert M Haralick, Stanley R Sternberg, and Xinhua Zhuang. Image Analysis Using Mathematical Morphology. *IEEE transactions on pattern analysis and machine intelligence*, (4):532–550, 1987.

[32] Navid Razmjooy, B Somayeh Mousavi, and Fazlollah Soleymani. A Real-Time Mathematical Computer Method For Potato Inspection Using Machine Vision. *Computers & Mathematics with Applications*, 63(1):268–279, 2012.

[33] Olivier Lezoray, Cyril Meurie, and Abderrahim Elmoataz. A graph approach to color mathematical morphology. In *Signal Processing and Information Technology, 2005. Proceedings of the Fifth IEEE International Symposium on*, pages 856–861. IEEE, 2005.

[34] Raman Maini and Himanshu Aggarwal. A Comprehensive Review Of Image Enhancement Techniques. *arXiv preprint arXiv:1003.4053*, 2010.

[35] P Rajavel. Image Dependent Brightness Preserving Histogram Equalization. *IEEE Transactions on Consumer Electronics*, 56(2):756–763, 2010.

[36] A. Walker R. Fisher, S. Perkins. Digital filters. `http://homepages.inf.ed.ac.uk/rbf/HIPR2/filtops.htm`, 2003. 2017-12-06.

[37] Kaiming He, Jian Sun, and Xiaoou Tang. Guided Image Filtering. *IEEE transactions on pattern analysis & machine intelligence*, (6):1397–1409, 2013.

[38] Utkarsh Sinha. Mathematical morphology: Dilation. `http://aishack.in/tutorials/mathematical-morphology/`, 2010. 2017-12-08.

[39] A. Walker R. Fisher, S. Perkins. Mathematical morphology: Dilation. `https://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm`, 2003. 2017-12-06.

[40] Wiki ImageJ. Particle counting in imagej. http://imagejdocu.tudor.lu/doku.php?id=video:analysis:particle_counting_-_automated_and_manual, January 2010. 2017-11-05.

[41] Mordvintsev Alexander. Opencv-python. http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_intro/py_intro.html#intro, 2013. 2018-08-10.

[42] Manzanera Antoine. Cours de morphologie mathématique. http://perso.ensta-paristech.fr/~manzaner/Cours/CentraleSupElec/MATIS4_Partie2_Papier.pdf, 2014.

[43] Github: Object counting. https://github.com/EmanuelOverflow/object-counting/blob/master/countobj.py, author = Emanuel, month = April, year = 2017, note = 2018-06-08.

[44] Web-based application. https://www.techopedia.com/definition/26002/web-based-application. 2018-08-05.

[45] Tj VanToll. What is the angular framework and why should developers use it? https://www.programmableweb.com/news/what-angular-framework-and-why-should-developers-use-it/analysis/2017/03/06, March 2017. 2018-07-30.

[46] Shamikh Hossain. What is flask framework used for in python? how is it different from django? https://www.quora.com/What-is-flask-framework-used-for-in-Python-How-is-it-different-from-Django-and-wh December 2017. 2018-07-30.

[47] Bsd licenses. https://en.wikipedia.org/wiki/BSD_licenses. 2018-07-30.

[48] angular file uploader. https://www.npmjs.com/package/angular-file-uploader, July 2018. 2018-07-10.

[49] Angular webstorage service. https://www.npmjs.com/package/angular-webstorage-service, December 2017. 2018-07-10.

Appendix A

Process development

Cells detection algorithm

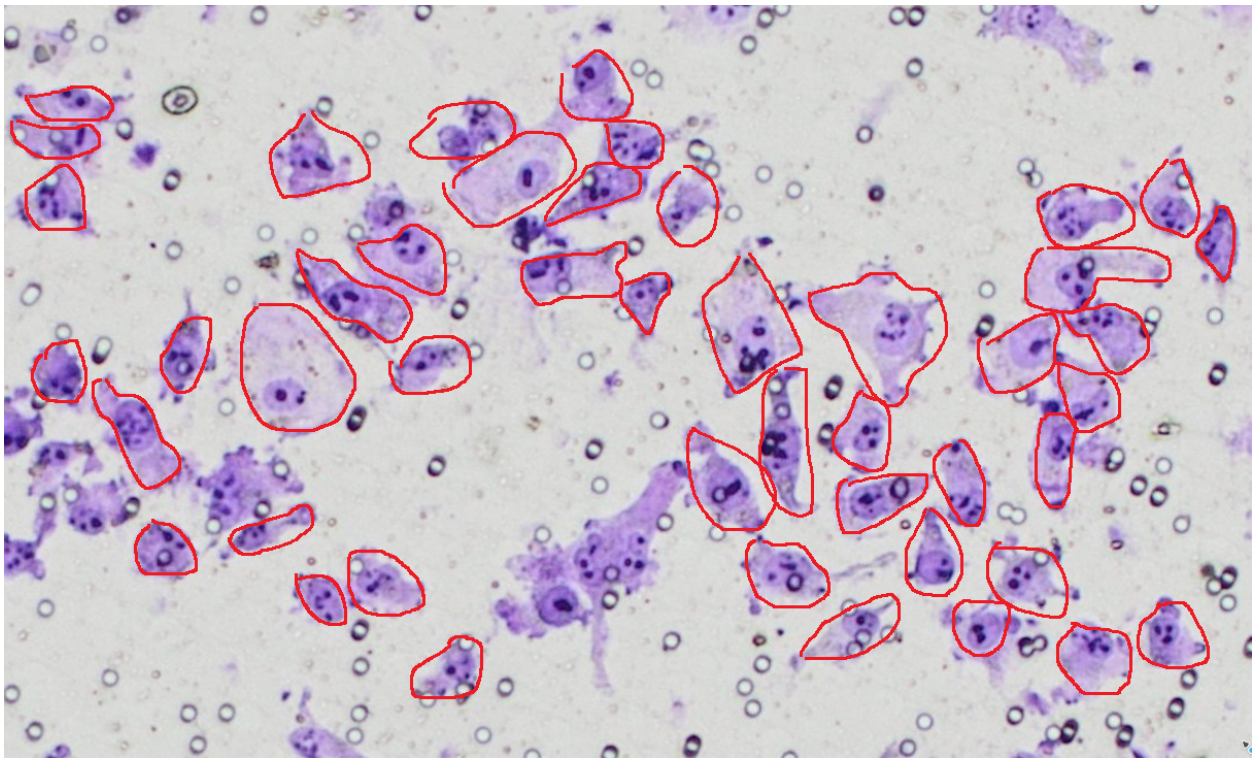


Figure A.1: Identification of cells by Marie Fourrez, technician in the 'Laboratoire de Biologie Moléculaire du Cancer'

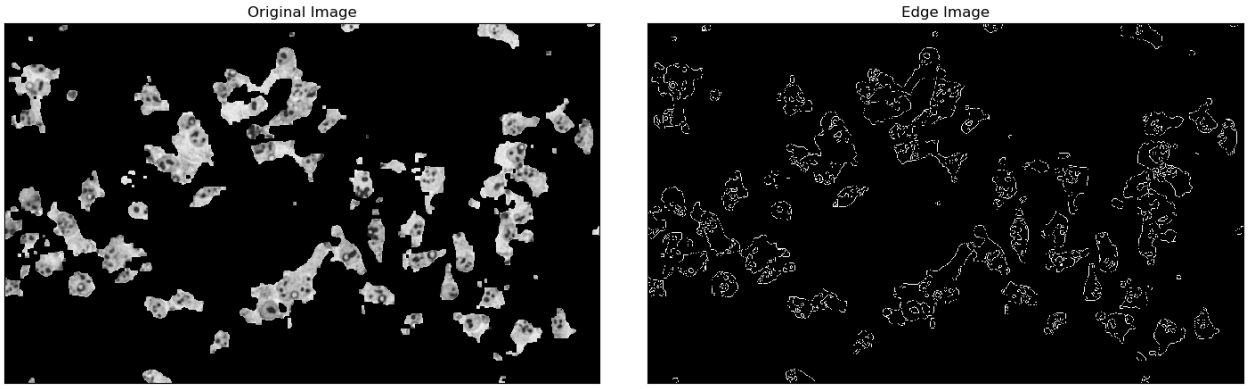
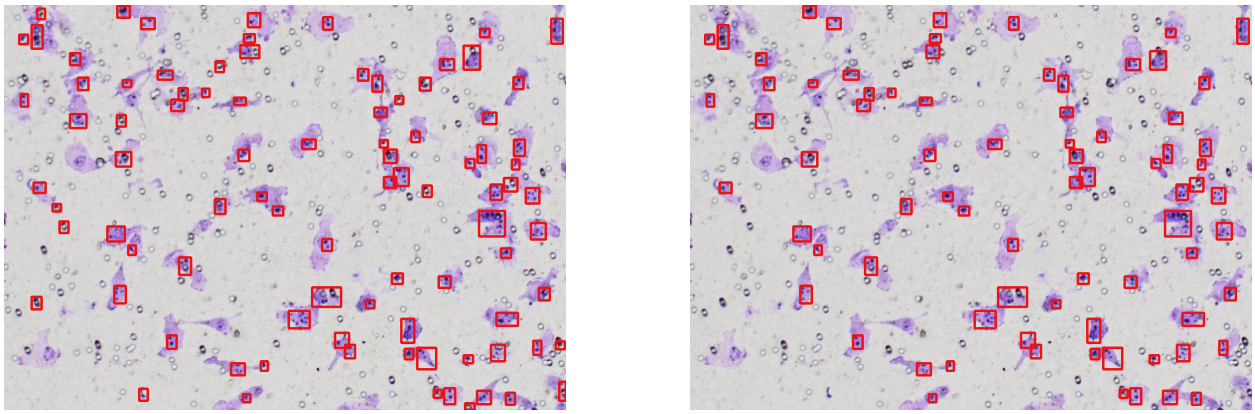


Figure A.2: Applying the edge detection on a restricted image does not give better results. The cells boundaries are not sufficiently clear.



(a) (b)

Figure A.3: Application of the cell enumeration on an other sample. Filter 4, erosion X, dilation 10 then filter 6, erosion 2 and dilation 12. In a), X = 4 and b) X = 5.
We can see that by this small adaptation, all what disappeared are pores.

```
import numpy as np
import cv2
import os
from processing.CellsCountPack import imgTools as tools

directory = os.path.dirname(os.path.abspath(__file__))
iteration = 6 #starting from 1

imIn = cv2.imread(directory + '/TargetCells.png')
imgHSV = cv2.cvtColor(imIn,cv2.COLOR_BGR2HSV) #HSV = hue sat
    value

hsv_min_values = [[125,50,0],[130,50,200], [125,50,0], [125,50,0
    ], [125,96,68], [90,34,0]]
hsv_max_values = [[145,255,255],[145,255,255], [145,200,255], [14
    5,255,255],[145, 255, 255], [145, 255, 255]]
min_purple = np.array(hsv_min_values[iteration-1])
max_purple = np.array(hsv_max_values[iteration-1])

mask = cv2.inRange(imgHSV, min_purple, max_purple)
res = cv2.bitwise_and(imIn, imIn, mask = mask)# there is
    something in the frame and the mask is true

cv2.imshow('frame',imgHSV)
cv2.imshow('mask',mask)
cv2.imshow('res',res)
cv2.imwrite('filter_result'+ str(iteration) + '.png', res)

img3 = tools.showMask(imIn, mask, True)
cv2.imshow('addition',img3)

### WRITE
#cv2.imwrite('filter_resultDiff'+ str(iteration) + '.png', img1)
#cv2.imwrite('filter_mask'+ str(iteration) + '.png', mask)

cv2.waitKey(0)
```

Figure A.4: NucleusIsolation.py

```
import cv2
import numpy as np
import os
from processing.CellsCountPack import imgTools as tool

directory = os.path.dirname(os.path.abspath(__file__))
f = '1'

original = cv2.imread(directory + '/TargetCells.png')
img = cv2.imread(directory + '/filter_mask'+ f+'.png', 0)
SE_erosion = 8
kernel_e = np.ones((SE_erosion,SE_erosion), np.uint8)
SE_dilatation = 10
kernel_d = np.ones((SE_dilatation,SE_dilatation), np.uint8)

img_erosion = cv2.erode(img, kernel_e, iterations=1)

cv2.imshow('Input', img)

resultErosion = tool.showMask(original,img_erosion, True)
cv2.imshow('Erosion', resultErosion)

img_dilation = cv2.dilate(img_erosion, kernel_d, iterations=1)
cv2.imshow('Dilation mask', img_dilation)
resultDilatation = tool.showMask(original,img_dilation, True)
cv2.imshow('Dilation', resultDilatation)

### WRITE
name_erosion = 'F'+f+'erosion_SE'+ str(SE_erosion) + '.png'
name_dilation = 'F'+f+'dilatation_motif'+ str(SE_erosion)+'_'
        + str(SE_dilatation) + '.png'
cv2.imwrite(name_erosion, resultErosion)
cv2.imwrite(name_dilation, resultDilatation)

cv2.waitKey(0)
```

Figure A.5: mathMorpho.py

```
import sys
import cv2
import numpy as np
import os

directory = os.path.dirname(os.path.abspath(__file__))

mask = cv2.imread(directory + '/dilation_mask8.png', 0)
original = cv2.imread(directory + '/TargetCells.png')
background = original.copy()

ROOT_NODE = -1

imgt = cv2.morphologyEx(mask, cv2.MORPH_OPEN, (5, 5))
img2 = imgt.copy()
contours, hierarchy = cv2.findContours(img2, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)
h = hierarchy
area = cv2.contourArea(contours[2])
print area

## Direct contouring #pass -1 if you want to draw all contours
# cv2.drawContours(background,contours,-1,(0,255,0),3)

fidelityRange = 200
totalContours = 0
br = []
for i in xrange(len(contours)):
    if h[0][i][3] == ROOT_NODE and
        cv2.contourArea(contours[i]) >= fidelityRange:
                totalContours += 1
                approx = cv2.approxPolyDP(contours[i], 3, True)
                br.append(cv2.boundingRect(approx))
for b in br:
    cv2.rectangle(background, (b[0], b[1]), (b[0] + b[2], b[1] + b[3])
        , (22, 20, 234), 3)
cv2.imshow('image',background)

print totalContours
cv2.imwrite('detection_fidelity-'+str(fidelityRange)+'.png',background)
cv2.waitKey(0)
```

Figure A.6: enumaration.py

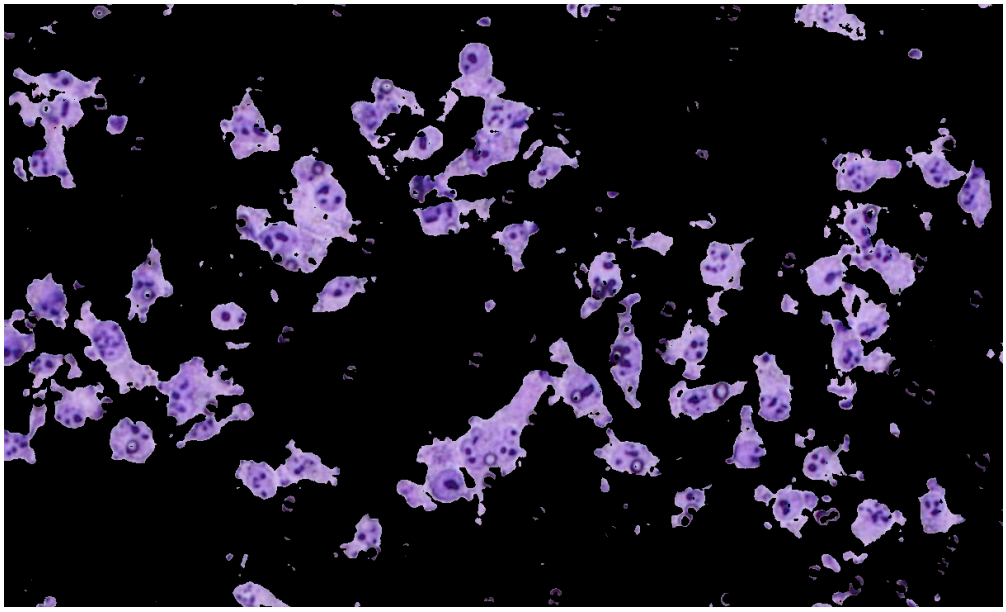


Figure A.7: Step 1: Original image restricted with a first filter of [125,50,0] - [145,255,255]

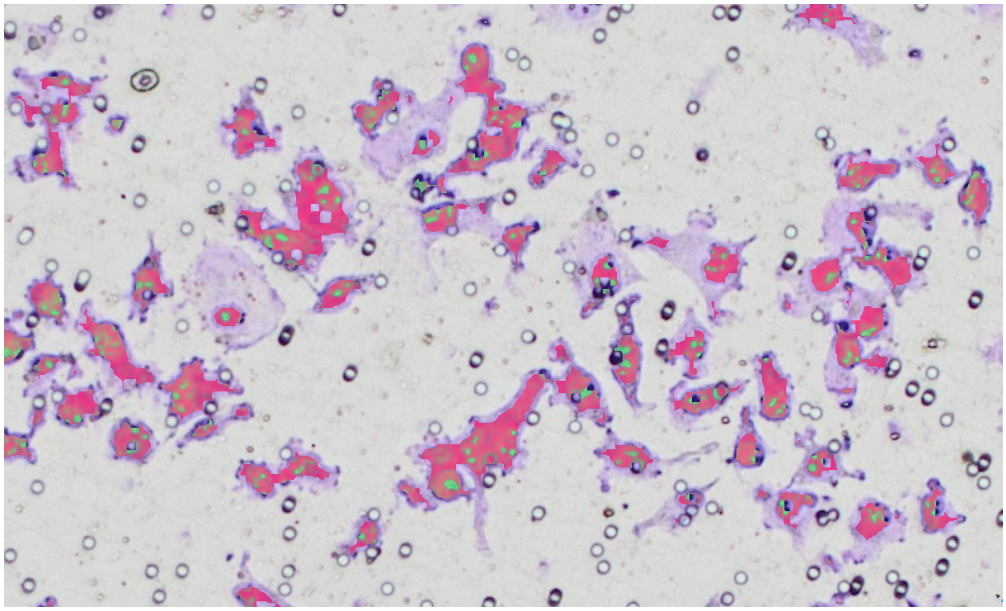


Figure A.8: Step 2: Erosion with SE of 9 on the mask obtained from the color filter to clean pores. The result is superimposed on the original image.

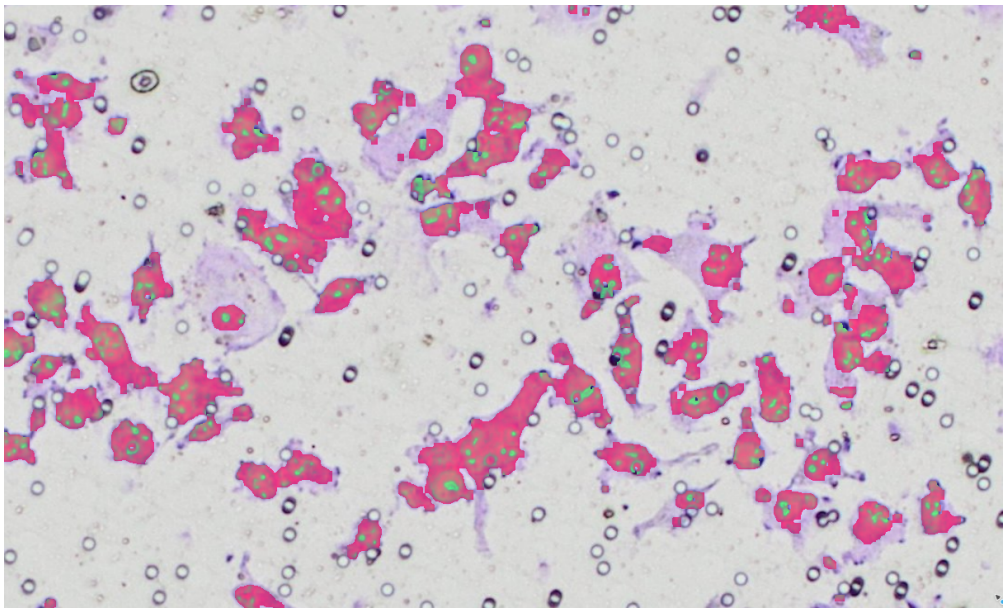


Figure A.9: Step 3: Dilation with SE of 8 on the mask already eroded to restore the volumes. The result is superimposed on the original image.

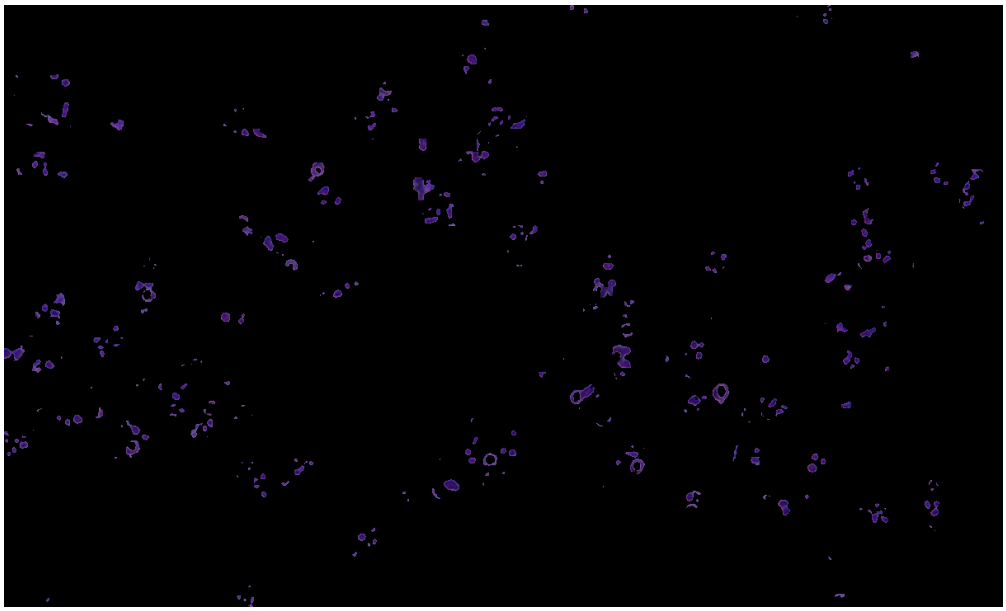


Figure A.10: Step 4: Second filter of [65,106,0] - [160,255,163] to isolate only the darkest elements, likely to be nucleoli.

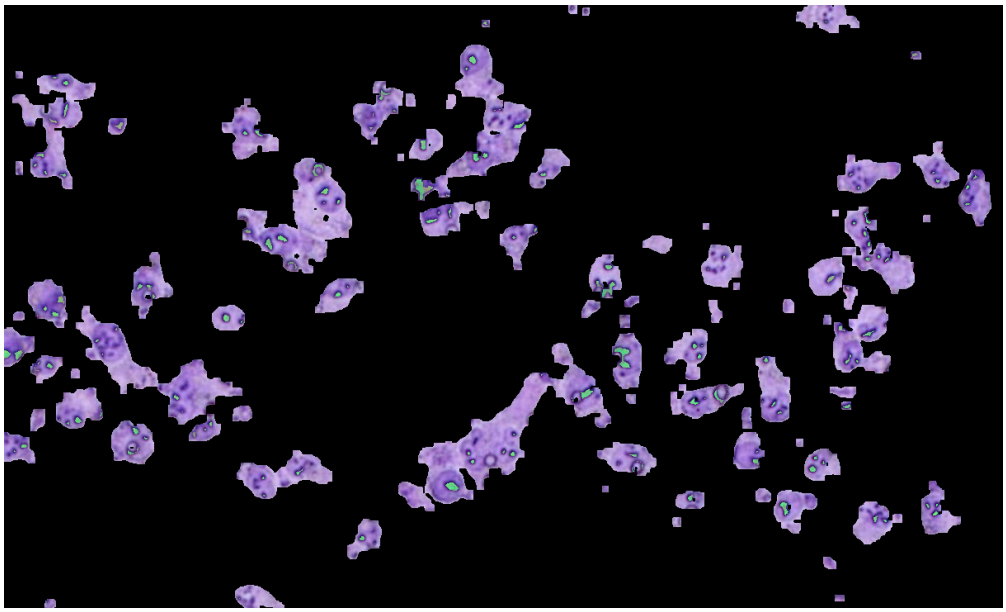


Figure A.11: Step 5: Erosion with SE of 4 on the mask obtained from the color filter to clean tiny isolated elements. The result is superimposed on the restricted image.

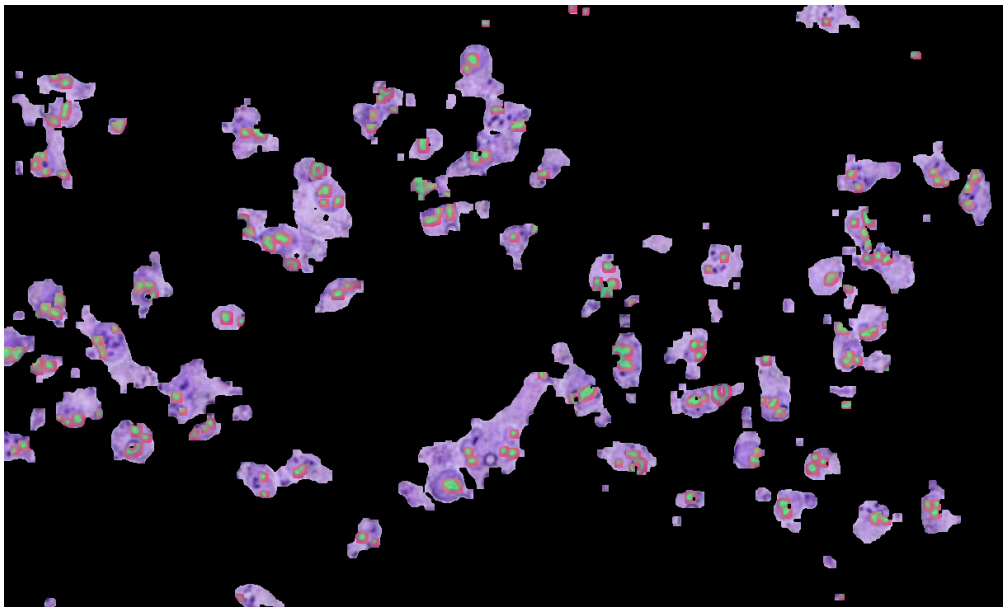


Figure A.12: Step 6: Dilation with SE of 10 on the mask already eroded to merge nucleoli together. The result is superimposed on the restricted image.

Graphical Interface

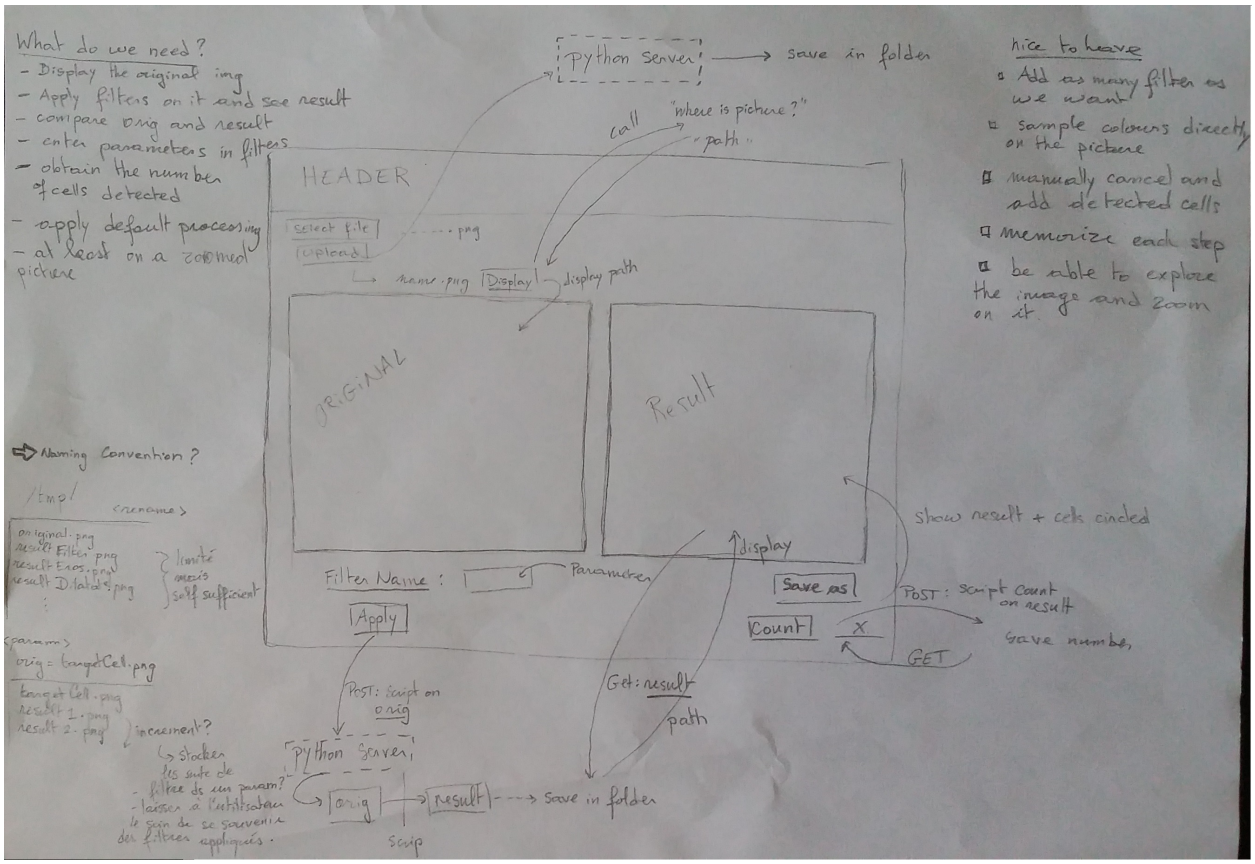


Figure A.13: Exploration sketch for the interface and exchange with the server.

```
# import the appJar library
from appJar import gui
import processingTools as PTools

def press(name):
    if name == "Cancel":
        app.stop()
    elif name == "Reset":
        app.clearEntry("Username")
        app.clearEntry("Password")
        app.setFocus("Username")
    elif name == "Submit":
        username = app.getEntry("Username")
        password = app.getEntry("Password")
        if username == "alitean" and password == "123":
            app.setImage("Light", "bulb_on.gif")
            app.infoBox("Success", "Valid password")

        else:
            app.setImage("Light", "bulb_off.gif")
            app.infoBox("Error", "Invalid password")
            app.disableButton("Cancel")

def transform(name):
    if name == "Cancel":
        app.stop()
    elif name == "Erode":
        app.setFocus("Motif")
        app.infoBox("Success", "apply erosion")
    elif name == "Dilate":
        motif = app.getEntry("Motif")
        if int(motif) > 0:
            app.setFocus("Motif")
            app.infoBox("Success", "apply dilation")
        else:
            app.infoBox("Error", "Invalid motif")
            app.disableButton("Cancel")
```

Figure A.14: CellCountApp.py

```
def exit():
    app.stop()

def openImg():
    global file
    file = app.openBox("Images", "/home/arianelit/Downloads/", [('images'
        , ('images', '*.png'), ('images', '*.jpg'), ('images', '*.jpeg')]
    app.setImage("img", file)

def filter():
    min_val, max_val = PTools.filterViewer()
    print min_val[1]
    new_limits = str(min_val) + "," + str(max_val)
    app.setLabel("limits", new_limits)
    filter_result = PTools.applyFilter(min_val,max_val)
    app.setImage("result", filter_result)
    app.zoomImage("result", -2)

app = gui("Login")
app.setResizable(canResize=True)
app.addLabel("lab1","Loading Window")
app.setLabelBg("lab1", "green")
app.setLabelFg("lab1", "white")
app.setFont(16)

app.addLabel("l1", "Load your cell image")
app.addButton("Open", openImg)

app.addImage("img", "TargetCells.png")
app.zoomImage("img", -2)
app.addLabelEntry("Motif")
app.addButtons(["Erode", "Dilate", "Cancel"], transform)

app.addButton("Exit", exit)
app.addButton("Viewer", filter)
app.addLabel('limits', "[0,0,0], [179,255,255]")
app.addImage("result", "TargetCells.png")
app.zoomImage("result", -2)

app.go()
```

Figure A.15: CellCountApp.py

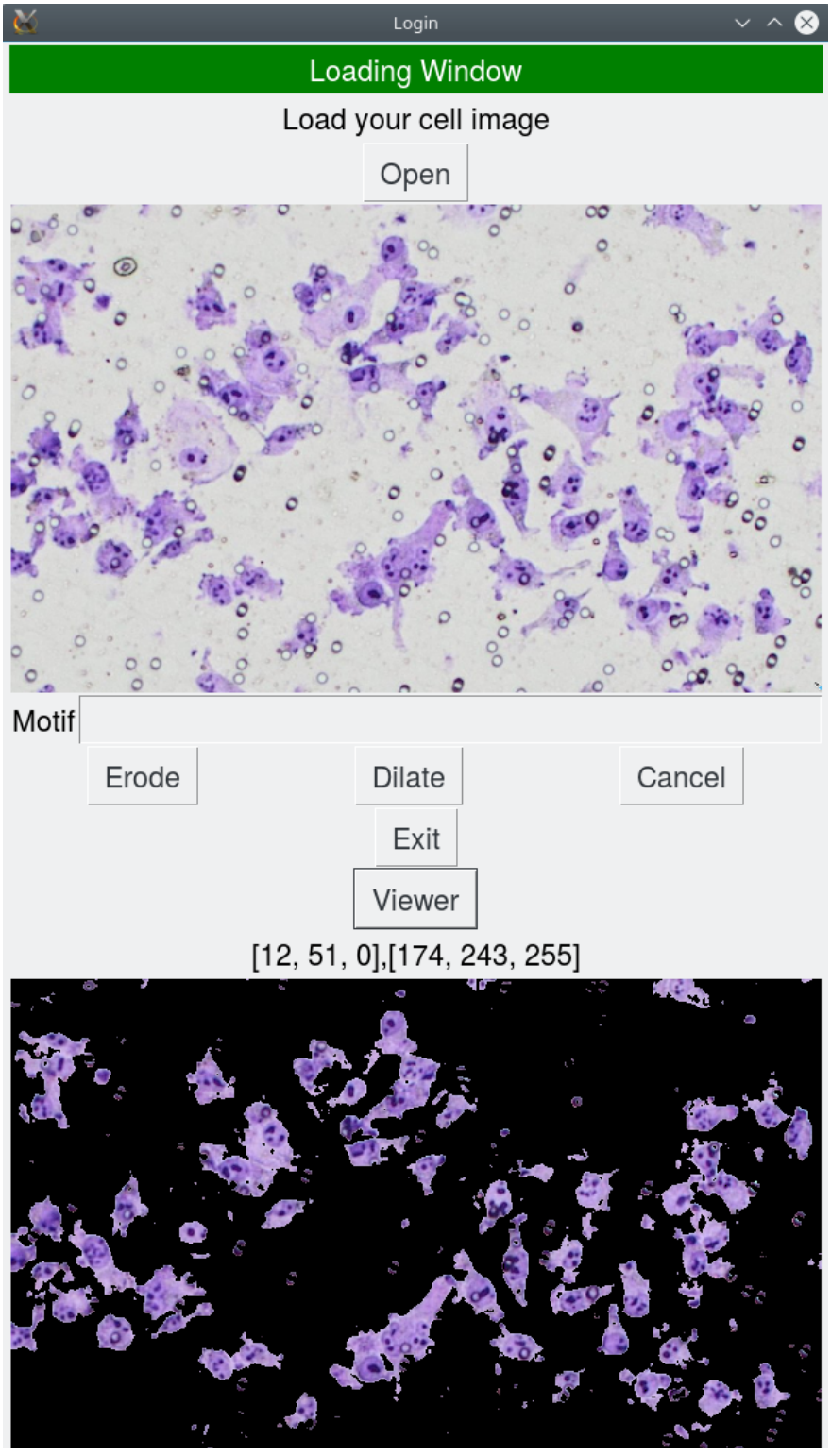


Figure A.16: Screen shot from the first page of the cell counting application with appJar

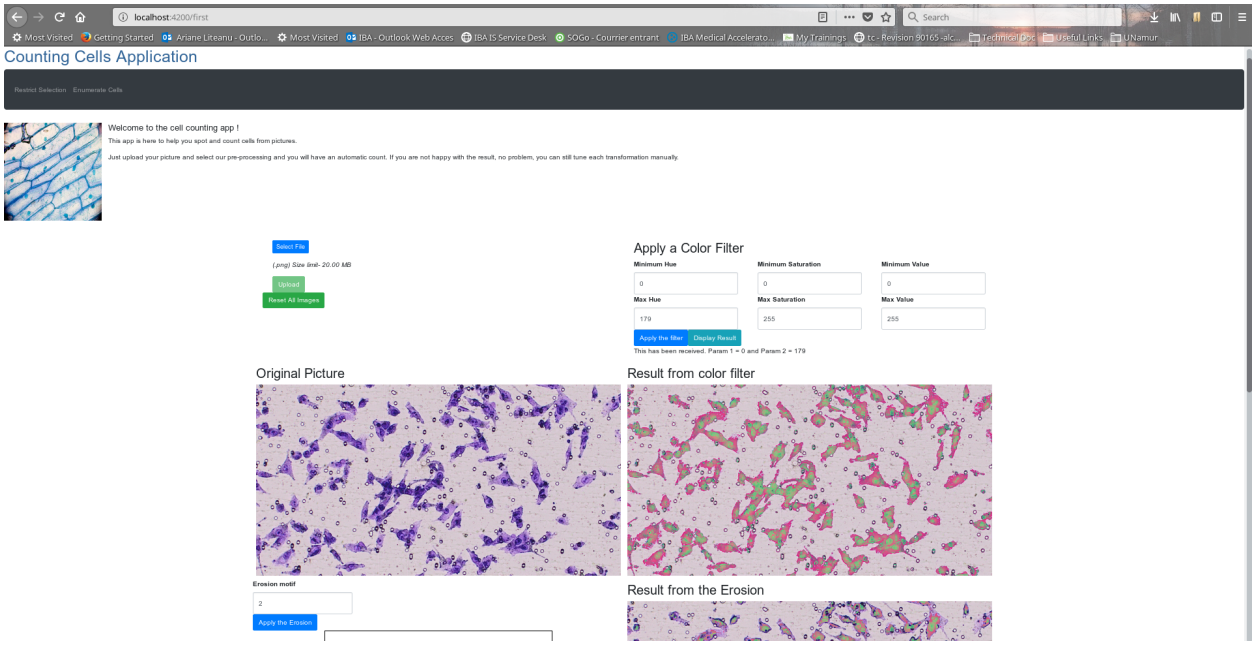


Figure A.17: Screen shot from the first page of the cell counting application with angular

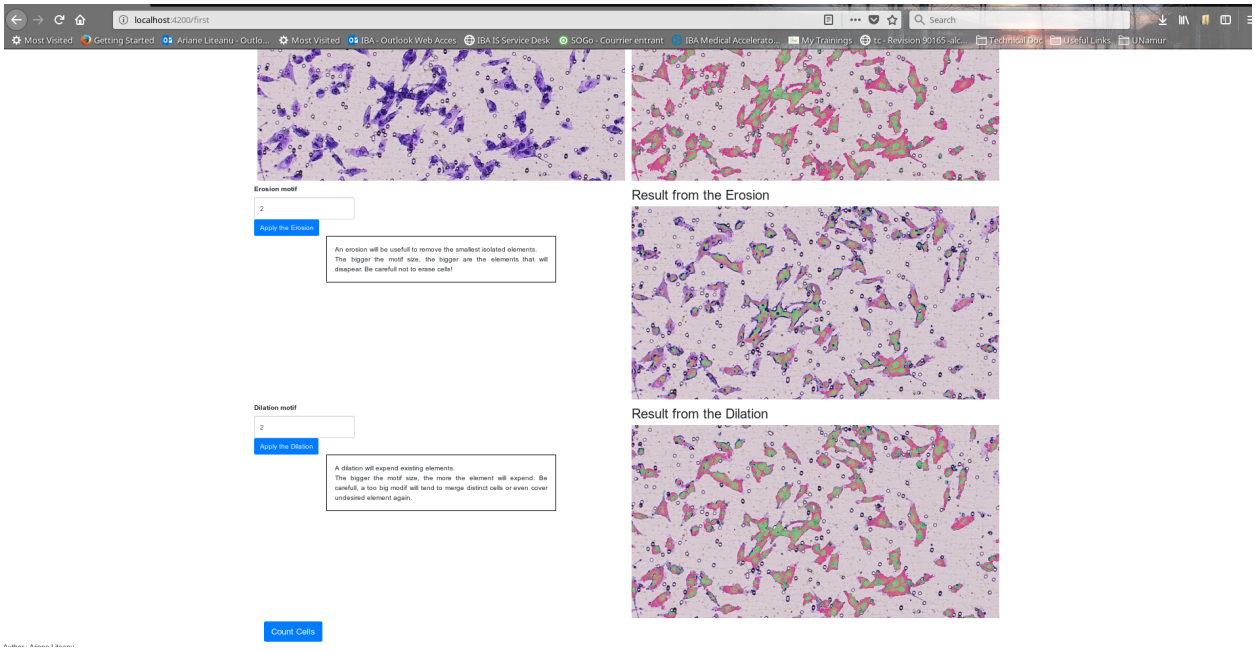


Figure A.18: First page of angular application.

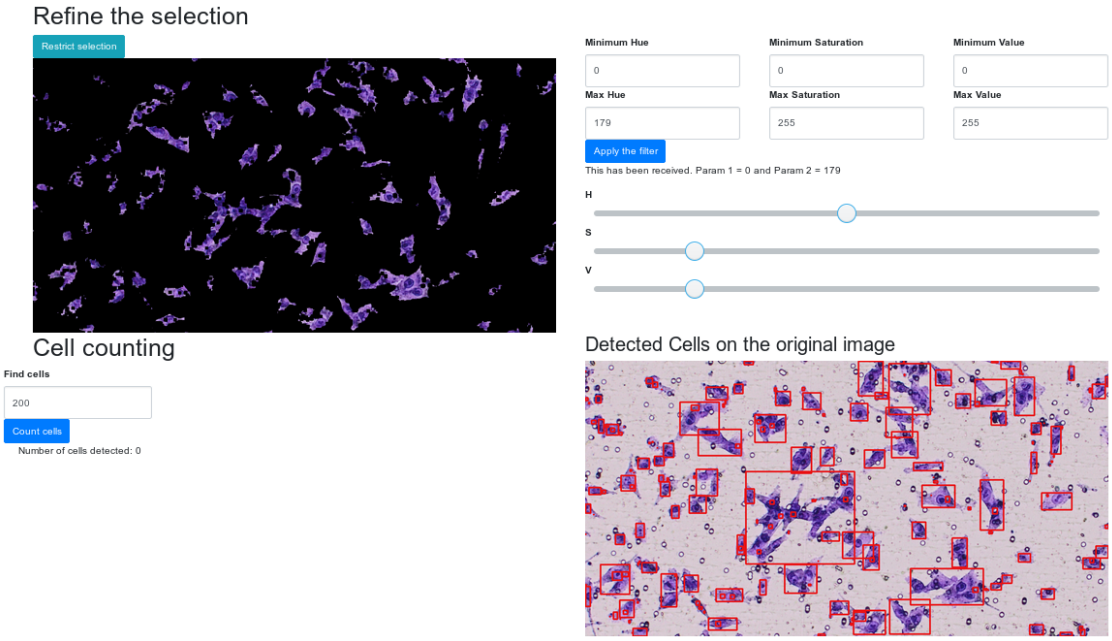


Figure A.19: Screen shot from the second page of the cell counting application with angular. The user can apply a second filter and launch the automatic enumeration.